Queue Mining:
Service Perspectives in Process Mining

Research Thesis

Arik Senderovich

Queue Mining:
Service Perspectives in Process Mining

Research Thesis

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Arik Senderovich

Submitted to the Senate of the
Technion - Israel Institute of Technology

Elul, 5776 Haifa September, 2016

This Research Thesis Was Done Under the Joint Supervision of
Professors Avigdor Gal and Avishai Mandelbaum in the Faculty of Industrial
Engineering and Management.

Acknowledgment

The generous financial help of the Technion is gratefully acknowledged. Special thanks to the faculty of Industrial Engineering and Management for the two excellence fellowships I received during the PhD.

To my supervisors. Avi, I am thankful for your dedication and support, you have been my mentor and guide throughout the degree. Our coffee time was priceless to advance our research, resolve personal issues, and discuss career opportunities I had a great time wherever we met and went: the office, Anna's place, and the plane. I am looking forward to having more of these in the future.

Avishai, thank you for introducing me to the fascinating world of research, data science, and services. The end of my PhD summarizes more than ten years of mutual work. It all started with a random chat in the hallway. In hindsight, that meeting, ten years ago, was a turning point that changed my life for the best, and for that I am truly grateful. I hope to continue working with you on joint research ideas for which we share our interests and passion.

Matthias, although not being my formal supervisor, you have been my de facto co-supervisor. Your ability to abstract and extract only the essence from things is admirable. I am thankful that I had (and still have) the chance to work with you and learn from you.

To my reviewers Roi Reichart and Isaac Meilijson, I am thankful for sharing your thoughts and feedback with me during the examination. To my many co-authors and colleagues. Andreas, it has been a true pleasure working with you. Quickly, we became friends, and I am sure that the connection we made, both personally and professionally, will last beyond this PhD work. To Alexander Shleyfman, Francois Schnitzler, Prof. Jan Mendling, Prof. Wil van der Aalst, Prof. Liron Yedidsion, Dr. Craig Bunnell, Sarah Kadish, Ryan Leib, Shahar Harel and Sander Leemans. Thank you. Your help, dedication, insights and hard work were important lessons and significant contributions to this PhD work. To my friends, Alex Shleyfman and Nitzan Carmeli, I am thankful for our discussions and your willingness to share and listen at all times.

To SEELab members Ella, Valery and Igor. I truly appreciate your help. I believe that I would miss some of my deadlines without your continuous support. Personally, I enjoyed being your SEENeighbor for the past four and a half years.

To my parents Irina and Yan, as well as my grandparents, Sarah and Lev. All this would not be possible without your endless support, from the early start and until now. Last, but most importantly, to my loving family, Liya, Loren and Alice - you are my everything. This thesis is dedicated to you.

Contents

Abstract

Business processes are supported by information systems that record process-related events in event logs. *Process mining* is a maturing research field that aims at discovering useful information about the business process from these event logs. Process mining can be viewed as the link that connects process analysis fields (e.g. business process management and operations research) to data analysis fields (e.g. machine learning and data mining).

This thesis relates to process mining techniques that aim at answering operational questions such as 'does the executed process as observed in the event log correspond to what was planned?', 'how long will it take for a running case to finish?' and 'how should resource capacity or staffing levels change to improve the process with respect to some cost criteria?

Prior to this thesis, process mining techniques overlooked dependencies between cases when answering such operational questions. For example, state-of-the-art methods for predicting remaining times of running cases considered only historical data of the case itself, while the interactions among cases (e.g. through queueing for shared resources) were neglected. The independence assumption is plausible in processes where capacity always exceeds demand. However, in service processes, which are prevalent in numerous application domains (e.g., healthcare, banking, transportation), multiple customer-resource interactions occur, and customers often compete over scarce resources. Consequently, the central argument of this thesis is that for service-oriented processes, process mining solutions *must* consider case interactions when answering operational questions.

The main contribution of this research thesis is the start of bridging a noticeable gap between process mining and queueing theory. To this end, we introduce *queue mining* (a term coined in this thesis), which is a set of data-driven methods (models and algorithms) for queueing analysis of business processes.

We demonstrate that considering the queueing perspective yields improved solutions to operational problems and enables solutions to problems that were not addressed earlier in the literature. Our queue mining techniques address the problems of prediction (delays and total times in the process), conformance to schedule (planned vs. actual), and process improvement (via production policy optimization). We demonstrate the effectiveness of these techniques with experiments on real-world data that comes from three domains: banking (a bank's call center), transportation (city buses), and healthcare (an outpatient hospital).

Chapter 1

Introduction

Modern business processes are supported by information systems that record process-related events in event logs. *Process mining* is a maturing research field that aims at discovering useful information about the business process from these event logs [66]. Process mining can be viewed as the link that connects process analysis fields (e.g. business process management and operations research) to data analysis fields (e.g. machine learning and data mining) [67].

This thesis is mainly concerned with process mining techniques that aim at answering operational questions such as 'does the executed process as observed in the event log correspond to what was planned?', 'how long will it take for a running case to finish?' and 'how should resource capacity or staffing levels change to improve the process with respect to some cost criteria? [66, Ch. 7, Ch. 8]. We refer to process mining solutions of such questions as *operational process mining*. Other types of process mining methods, such as social network mining and process model visualization are beyond the scope of the present work.

To the best of our knowledge, and prior to this thesis, process mining techniques overlooked dependencies between cases when answering such operational questions. (We shall refer to process instances, cases, customers and jobs throughout the thesis interchangeably.) For example, state-of-the-art methods for predicting remaining times of running cases considered only historical data of the case itself [71], while the interactions among cases (e.g. through queueing for shared resources) were neglected. The independence assumption is plausible in processes where capacity always exceeds demand. However, in service processes, which are prevalent in numerous application domains (e.g., healthcare, banking, transportation), multiple customer-resource interactions occur, and customers often compete over scarce resources. Consequently, the central argument of this thesis is that for service-oriented processes, process mining solutions *must* consider case interactions when answering

operational questions.

The main contribution of this research thesis is the start of bridging a noticeable gap between process mining and queueing theory. To this end, we introduce *queue mining* (a term coined in this thesis), which is a set of data-driven methods (models and algorithms) for queueing analysis of business processes. The methods that are used in queue mining are described in Chapter 2.

In our Findings chapter, Chapter 3, we demonstrate that considering the queueing perspective yields improved solutions to operational problems and enables solutions to problems have not been addressed earlier in the literature. Our queue mining techniques address the problems of prediction (delays and total times in the process), conformance to schedule (planned vs. actual), and process improvement (via production policy optimization). We demonstrate the effectiveness of these techniques with experiments on real-world data that comes from three domains: banking (a bank's call center), transportation (city buses), and healthcare (an outpatient hospital).

The remainder of this introductory chapter starts with an overview of the process mining field (Section 1.1) with special focus on operational process mining where we emphasize gaps in the existing literature. Then, we outline the main contributions of our research (Section 1.3) and conclude the chapter with describing the thesis structure (Section 1.4).

1.1 Process Mining

Process mining is traditionally perceived to cover three main mining problems, specifically *discovery*, *conformance checking* and *model enhancement*.

Discovery. The first problem that process mining aimed historically at solving was discovery of control-flow models from event logs [66, Ch. 5,6]. An example for a control-flow model is presented in Figure 1.1; the process model describes patient flow at the Dana-Farber Cancer Institute (DFCI), a large outpatient cancer hospital in the United States. (The modeling language for Figure 1.1 is Business Process Model and Notation (BPMN) [47].) Patients undergo a sequence of activities that typically comprise of blood draw, physician's examination, and a chemotherapy treatment (see Chapter 3 for more details about this process). The control-flow perspective captures the main activities, routing decisions (e.g. probabilistic splits, forks), exceptions, etc.

The seminal work by van der Aalst et al. [68] proposed the first (control-flow) discovery algorithm, named the α-algorithm [68]. The α-algorithm maps event logs into workflow-nets (WF-Nets) [65], which are special cases of the Petri net formalism [9]. The α-algorithm

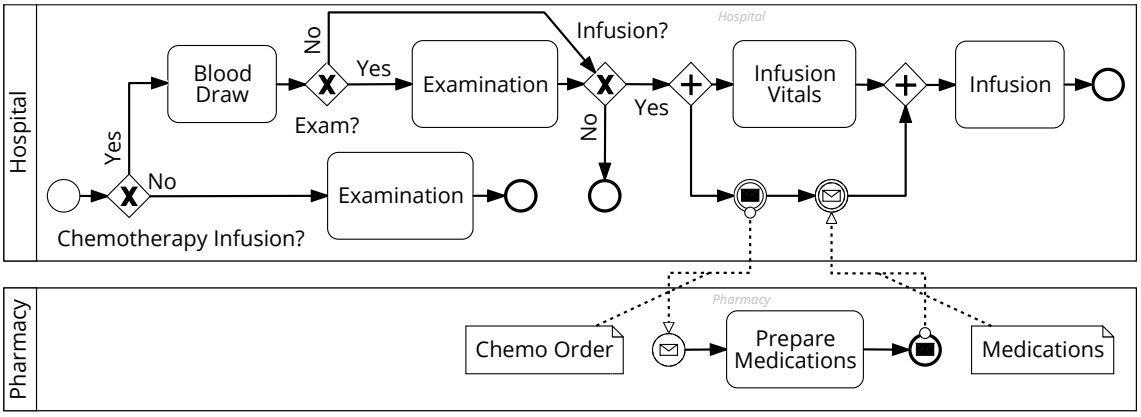


Figure 1.1: Conceptual control-flow model of patient flow at the Dana-Farber Cancer Institute (DFCI) in Boston, USA.

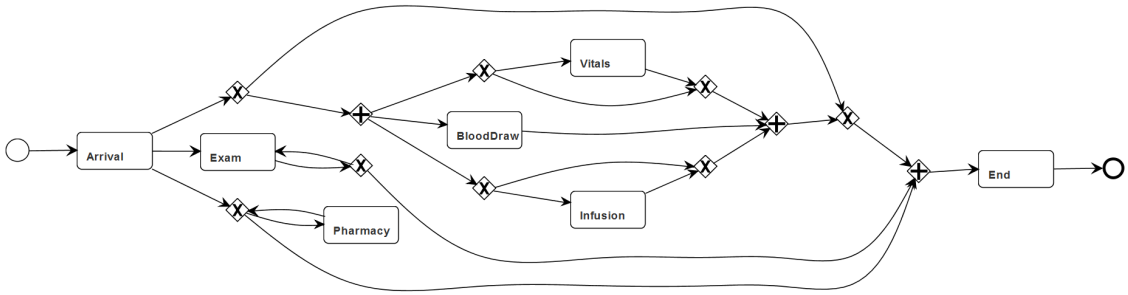


Figure 1.2: Data-driven model of patient flow at DFCI.

and other early algorithms were limited in their applicability to real-life event logs due to several unrealistic assumptions. For example, event logs were assumed to be noise free (no erroneous recordings) and complete (every possible behavior was recorded in the event log). These assumptions rarely hold in real-world event logs (see [66, Ch. 5] for a comprehensive list of limitations). In practice, these limitations result in over-complex process models with multiple structural flaws.

Consequently, advanced process discovery algorithms were developed to overcome these limitations. A representative selection of advanced discovery algorithms includes the *heuristic miner* [76, 77], the *fuzzy mining* algorithm [31], and the *genetic mining* algorithm [23]. Recently, well-formed Petri nets, aka block-structured Petri nets, were proposed as a new representation for discovered process models [14, 39]. Figure 1.2 presents a well-structured (control-flow) model automatically discovered by the Inductive Miner from the Dana-Farber Cancer Institute data [39]. Note that Figure 1.2 can be viewed as the data-driven counterpart

of Figure 1.1.

Discovered control-flow models cannot be used for operational analysis, since these models lack information regarding time and resources. Therefore, their main purpose is visualization and verification of the control-flow perspective. For example, DFCI management may use such models to get an evidence-based view of the patient flow. Further, it is often useful for visualizing anomalies such as deviating paths and skipped activities. This visualization provides an empirical spotlight on problems in the current process. For recent advances in automated process discovery, the reader is referred to the survey part of [24].

Conformance Checking. After a process model is available (either discovered from data or being a normative model provided by a human modeler), there is often the need to check whether the model and its log are aligned. The verification of alignment between a given model and an event log is referred to as conformance checking [66, Ch. 7]. For normative models, deviations may indicate that the real process does not follow guidelines and business rules [3, 49, 69] or alternatively, that the event logs contain erroneous information. For data-driven models, the alignment returns the distance between the resulting model and the log, thus quantifying the level of model abstraction with respect to the event log. Discovered models that fit the log have a low abstraction level (and thus do not generalize), while highly abstract models will deviate from reality, as it was recorded in the data, by allowing behavior that is not observed in the log.

The first technique for conformance checking was proposed in [54]; it is based on the concept of *replay*. The idea behind replay is that the event log is 'replayed' on-top of the process model (e.g., consider the token game for Petri nets, with event data being the tokens); correspondences between moves in the log and possible moves in the model indicate conformance, whereas discrepancies between the two provide evidence for non-conformance. There are two discrepancy types that can be captured by replay: (1) sequences in the log that cannot be executed according to the model, and (2) behavior that is possible in the model and was not observed in the event log.

The replay technique was recently extended to an optimization setting that results in an *optimal alignment* between model and log [2]. Here, the two types of deviations are assigned two corresponding costs. One cost represents the importance of finding recordings in the event log that are not allowed according to the model, while the other cost corresponds to parts of the model that do not appear in the log. Then, an optimal alignment that reduces deviation's costs between the log and the model is calculated.

Two notable process mining techniques that are based on conformance checking are *model repair* and *model simplification*. In model repair, it is assumed that the event log

records correct executions of the business process, while the model (conceptual or discovered) deviates from the log. The deviations are detected via a conformance checking algorithm. Then, the non-conforming model is modified (iteratively) until it converges to a conforming model [11, 26]. In model simplification, an overfitting model is mined from the event log. Then, to improve generalization, the model is simplified, which makes it less sensitive to infrequent behavior and thus avoid overfitting [25]. For a comprehensive survey on conformance checking and its applications see [66, Ch. 7] and [75].

Conformance checking for additional dimensions beyond the control-flow, such as time and resources, is essential when aiming at answering operational questions on the alignment of models and logs [64]. Therefore, in order to answer operational questions, control-flow models must be enhanced with these additional perspectives.

Model Enhancement. Model enhancement takes us one step closer to operational process mining. Specifically, enhancement algorithms use event logs that include additional information, e.g., execution times, resources, and activity costs, to transform control-flow models into operational models. Specifically, in [51] the authors enrich Petri nets into their timed counterparts to answer prediction queries. Further, in [55] Petri net models are enhanced with decisions, resources, and times to create simulation nets. Some of the additional perspectives that were considered in model enhancement literature include the time perspective [51, 71, 73], the organizational perspective [70] (mining social networks), the case perspective [53] via decision mining, the resource perspective (resource-aware process analysis [46]), and multi-dimensional conformance checking [21, 43]. Enhancement techniques that involve time and resources can be viewed as *operational process mining*.

Bringing It All Together. As process mining has matured into a research field, the three tasks (i.e., discovery, conformance, and enhancement) have been integrated into a single methodology for data-driven process analysis [45, 67, 74]. The concept behind this holistic approach is as follows. An event log is used to discover a model that includes operational perspectives (i.e., a combination of discovery and enhancement). Then, the model is compared to the event log (conformance checking) and the model is repaired accordingly (in case of a non-conforming model). Further, the conformance checking step (for different perspectives) can already answer several operational questions ('where did the process deviate from what was planned?'). Lastly, the conforming (multi-perspective) model is applied to answer questions on performance, e.g., time prediction, capacity analysis and process improvement [57]. In this thesis, we adopt this holistic approach for operational process mining.

1.2 Operational Process Mining

In this section, we provide a literature survey of operational process mining.[1] Specifically, we go over three streams of work, namely *simulation mining*, *supervised process mining*, and *multi-perspective conformance checking*. The former two lines of work focus on performance-oriented operational questions, while the latter focuses on the alignment between model and log. Towards the end of this section, Table 1.1 provides a summary of the current literature across several dimensions.

Simulation Mining. Early works on performance-oriented process mining were based on the enhancement of control-flow models with further data-based information such as decisions, resources and time [55]. The work of Măruşter and van Beest [44] extends this approach from modeling and analysis of an existing process to its improvement based on the discovered model. Due to the richness of these discovered models they can only be analyzed via simulation. Thus, we refer to these methods as simulation mining or model-based process mining, interchangeably.

The main benefit of these model-based techniques is their ability to solve a large set of performance problems. For example, one can use simulation to predict waiting times and propose resource staffing. However, current simulation mining techniques overlook queueing effects by simulating process cases independently of each other and drawing waiting times (and other measures that stem from case dependencies) from fitted distributions. Furthermore, model-based mining techniques tend to overfit reality as it is observed in the event log and therefore it was shown that simpler performance models often provide accurate results and are more efficient [61]. The expressiveness problem is partially addressed in [52], where authors mine non-Markovian stochastic Petri nets [32], which are simple formalisms that account for control-flow and exponential durations. In [61], the best of both worlds is combined by a method for simulation mining that captures queueing effects and simplifies the resulting model. This simplification results both in improved efficiency (run-time complexity of the solution) and accuracy.[2]

Supervised Process Mining. Another prevalent stream of work in process mining is the encoding of event logs into well-established supervised learning methods. Examples for such methods can be found in [22, 40, 48, 71]. In these works, both the behavior and the operational aspects of the process are encoded: the former representing the control-flow

[1] Solving operational problems based on data is not limited to process mining. For references to works outside the process mining discipline, see [52] (for operations research), and [29] (for data analysis).

[2] The work is a result of the current research. It is not included in this thesis.

perspective and the latter capturing activity durations and resources. While these approaches are often accurate in solving a given mining problem with respect to a set of performance measures observed in the event log (e.g. total time in process), it has two major drawbacks.

First, one cannot use these techniques for quantifying measures that are not directly observable in the log. For example, information that allows one to calculate resource utilization might be missing from the event log, and thus supervised approaches cannot be applied. Second, exploring process improvement directions and sensitivity of process parameters (e.g. durations and arrival rates) is impossible due to the lack of data that describes the effect of changing these parameters. In [29] and [62], we propose supervised process mining solutions that bring together the benefits of model-based mining (or simulation mining) and supervised learning.[3] Specifically, these solutions comprise machine learning methods (e.g. decision trees) and queueing models. This combination enables quantifying measures that were not observed in the event log due to the existence of the model. For measures that appear in the event log we get accurate solutions by exploiting the strengths of supervised learning [34]. Furthermore, we are able to have what-if reasoning based on the queueing model, without the need to acquire further data from the changed process.

Multi-Perspective Conformance Checking. In this thesis, we call operational process mining to works on multi-perspective conformance analysis that verify alignment of process models and event logs with respect to time and resources [20, 43]. The approach in the literature is to compare deterministic data values according to a given model to those available in the log. For example, activity durations are considered to be known values (according to the model) and the alignment procedure checks whether or not event log durations are equal to model durations. However, the reality in processes is often random (random durations, random arrival process, etc.). Therefore, a methodology for comparing distributions (that come from the model) and deterministic values (coming from the log) is required. We show such an approach in our work on conformance checking from the queueing perspective [64] (the work is presented in Chapter 3). Further, our work proposes a methodology that considers queueing interactions that were neglected in existing methods, and recommends improvement based on the result of the conformance checking procedure [64].

Summary. Table 1.1 provides a qualitative summary of the operational process mining literature along the following dimensions:

- Measure Variety: This dimension indicates the richness of measures that the solution handles. For example, if a technique is able to learn and predict a single measure (e.g.

[3] These works appear in Chapter 3.

Work Stream	Measure Variety	Problem Range	To-be Analysis	Efficiency	Queueing Persp.
Simulation mining [44, 52, 55]	+	+	+	-	-
Supervised process mining [22, 27, 40, 48, 71]	-	+	-	-	-
Multi-perspective conformance [19, 20, 21, 43, 72]	+	-	-	-	-
Queue mining (this thesis) [29, 60, 61, 62, 64]	+	+	+	+	+

Table 1.1: Operational process mining literature.

resource delays only) we consider it to have a narrow spectrum of measures (and it will get a '-' in the table).

- Problem Range: The diversity of the problems that can be solved (e.g. prediction only vs. prediction, conformance, and process improvement).
- To-be analysis: The column indicates whether or not the technique accommodates process changes.
- Efficiency: The run-time complexity of the technique. We denote with + if the technique supports real-time analysis in reasonable amount of time.
- Queueing: Indicates whether the solution considers the queueing perspective. This aspect was not handled prior to the appearance of queue mining.

Table 1.1 clearly depicts the main contributions of this thesis.

1.3 Thesis Contributions

The main overall contribution of this thesis is the development of queue mining, which is a set of operational process mining solutions based on the discovery of queueing models from event data. These solutions explicitly capture delays due to case-resource interactions. Below, we detail the specific contributions of this research and outline their impact in two research fields, namely process mining and queueing theory (or, more broadly, operations research).

1.3.1 Contributions to Process Mining

The thesis makes the following contributions to process mining:

- The work integrates data-driven queueing modeling and analysis into operational process mining [29, 60, 61, 62, 63, 64], namely queue mining.
- Our work links operational conformance analysis to performance improvement in business processes [64]. Specifically, we develop a methodology for the detection of problematic areas in a process (e.g., due to synchronization or resource delays) and propose proper adjustments to queueing policies (selection of the next case to enter service) in these problematic parts.
- We make advances in the area of multi-dimensional conformance analysis, by adding the queueing perspective, structuring the approach, and demonstrating how conformance checking can lead to performance improvement [64].
- We propose hybrid methods that combine supervised learning and model-based process mining techniques to improve the accuracy of answering operational questions [29, 62]. This combination improves accuracy with respect to applying the two approaches separately.

Three of the works mentioned in the list above, [29, 62, 64] are part of Chapter 3. Other works were published in papers that were not included in the thesis.

The proposed methods were validated based on three real-world datasets coming from service domains. In [60, 62] we evaluate queue mining against a bank's call center; in [61, 63, 64] we mine the dataset of the Dana-Farber Cancer Institute (a large outpatient cancer hospital in the United States); and in [29] we evaluate our techniques on GPS bus data that comes from the city of Dublin.

1.3.2 Contributions to Queueing Theory

Queueing theory has been mainly a theoretical research field that aims at modeling and analyzing (typically) stochastic and dynamic service or manufacturing systems [17, 41]. Some works in queueing theory provide numerical experiments that justify the practicality of theoretical results. The underlying data is typically simulated based on systems that adhere to various assumptions that fit the models. Only a small sample of works in queueing theory include real-world data validation (see [5, 7, 42] and references within).

In this work, we validate multiple results from queueing theory, thus showing their practical applicability to real-world data. Further, we provide a platform for testing new and (other) existing results in queueing theory based on event data (as long as the event log is standardized according to our definition in Chapter 2, and the queueing model is

well-defined). Specifically, all our techniques were implemented as prototype systems that are available as open-source. For example, the methods that we use for delay prediction [62] can be extended, replaced, and tested against, when new prediction methods become available. In other words, this research creates a playground for queueing theorists to empirically evaluate their developed theories and results.

1.4 Structure of Thesis

The remainder of this thesis is structured as follows. In Chapter 2 we outline the methods that we use throughout the Findings chapter. Specifically, we outline the models that we use while developing queue mining algorithms, before providing an overview of queue mining methods. Chapter 3 presents the findings of this thesis, which comprises three journal papers. Lastly, in Chapter 4 we conclude the thesis with insights from our main findings.

Chapter 2

Methods

The methods that we develop and apply for solving problems in operational process mining are referred to as *queue mining*. In our context, queue mining comprises several prediction and conformance solutions (models and algorithms) that jointly cover different aspects of the dimensions mentioned in Table 1.1. Queue mining methods are based on two types of conceptual models: queueing models (Fork/Join networks) and data models (event logs). In this section, we start by formalizing Fork/Join networks and event logs. Then, we define operational mining problems and discuss queue mining techniques that solve these problems, with the focus being two approaches taken in this thesis, namely model-based and supervised learning.

2.1 Queueing Models

In this part, we introduce a formalization of queueing models. The discussion is primarily based on the definition of queueing models that appears in [64], and introduces a general type of queueing network, namely *Fork/Join (F/J) networks*.

2.1.1 Definition

Formally, queueing networks are directed graphs, with vertices corresponding to server nodes (or service providers). Fork/Join (F/J) networks support splitting and joining of customers, which makes them particularly suitable for modeling concurrent processing [8]. F/J networks support two types of queues: *resource queues* are formed due to limited resource capacity, and *synchronization queues* result from simultaneous processing by several resources. Servers can thus be of three types, namely (1) resource servers (corresponding to resources with finite or infinite capacity), (2) fork servers, and (3) join servers.

In this work, we consider *open* F/J networks (customers arrive from outside the system and depart eventually) that exhibit *multi-class* services (servers execute several types of activities), and *probabilistic choices*. This formalism is inspired by the conceptual framework of *processing networks* [1, 33], and generalizes both multi-class networks [37] and Fork/Join networks [8].

In multi-class queueing networks, customers that arrive to a server may encounter different types of processing. Examples for such types of processing include not only the execution of different activities, but also different ways of executing an activity. For instance, in scheduled processes, activities may be planned with different durations, so that each combination of an activity and its planned duration is treated differently in the scheduling of resources, and is considered to be a separate class.

As a concrete example, consider a physician that performs two types of activities—examination and consultation. Further, examinations can be planned for 15 minutes or 30 minutes and, depending on the duration, the scheduling is implemented differently. As such, in this example, there are three customer classes (consultation, examination in 15 minutes, examination in 30 minutes). Formally, this aspect is captured by a set of *customer classes*, which we denote by $C \subseteq \mathbb{N}^+$.

To define F/J networks, we need to specify *server dynamics* for each of the servers in the network. To this end, we adopt a version of Kendall's notation [38], so that every server (or station) is characterized by five building blocks, $\mathcal{A}_t/\mathcal{B}_c/\mathcal{R}_t/\mathcal{Z}/\mathcal{P}$: $\mathcal{A}_t$ represents the (time-varying) *external arrival process*, which is assumed to be class-independent; $\mathcal{B}_c$ corresponds to (stationary) *processing time distribution* for customer class $c \in C$ (time-independence is typically assumed for service processes, see [13]); and $\mathcal{R}_t$ stands for time-changing resource *capacity*, i.e., the number of resources working in the server node at time t. The *class assignment* function, $\mathcal{Z}$, assigns a class to an arriving customer with probability $Z(c)$, where $\sum_{c \in C} Z(c) = 1$.

Component $\mathcal{P}$ is the stationary *service policy* that sets both the order of entry-to-service, among the enqueued customers, and selects the resource, among available ones, to serve a customer. For example, the most common service policy for queues is the First-Come First-Served (FCFS) policy. Resources related to server nodes are work-conserving (immediately engaging in service when becoming idle and a customer is waiting) and statistically identical with respect to the distribution of their processing times. All five building blocks are assumed to be independent of one another.

Let $\mathcal{K}$ be the universe of possible model dynamics for a server; we define the F/J network as a probabilistic network of three different types of server nodes, each server being assigned a model of dynamics.

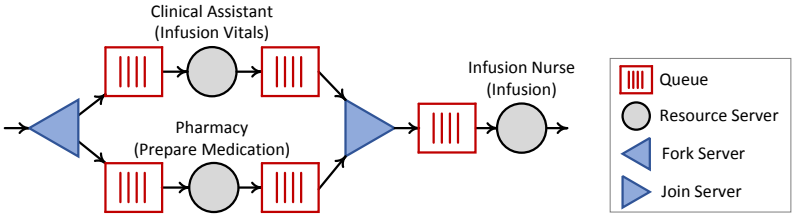


Figure 2.1: Patient flow in Dana-Farber Cancer Institute: A Queueing Perspective.

Definition 1 (Fork/Join Network)**.** *A* Fork/Join network $\mathcal{F}$ *is a triple* $\langle S, W, b \rangle$*, where*

- $S = S_R \cup S_F \cup S_J$ *is a set of servers, with* S_R *being a set of resource servers and* S_F, S_J *being sets of forks and joins, respectively;* S_R*,* S_F*,* S_J *are disjoint.*
- $W : (S \times S) \to [0, 1]$ *is a routing matrix (or the weighted flow) between servers;*
- $b : S \to \mathcal{K}$ *assigns a model of dynamics to servers.*

The weights of the routing between servers correspond to probabilities. Therefore, $0 \leqslant W(s, s') \leqslant 1, \forall s, s' \in S$ and $\sum_{s' \in S} W(s, s') = 1$. We consider forks and joins to be zero-delay and zero-capacity resource nodes. The weights of arcs coming out (in) of a fork s_f (a join s_j) are assumed binary, that is, a customer always routes to (from) a downstream (upstream) server, s', and thus $W(s_f, s') = 1$ ($W(s', s_j) = 1$).

Example. As an example, consider the F/J network in Figure 2.1, which depicts the queueing perspective for one part of the Dana-Farber Cancer Institute process (see Figure 1.1 for the control-flow perspective). This visualization contains, in addition to server nodes and routing, three resource queues (preceding resources) and two synchronization queues (succeeding resources). It is worth noting that an F/J network is *fully characterized* by servers, routing, and server dynamics. Queues are defined implicitly before and after their respective servers. The example, Figure 2.1, contains three resource types, a fork, and a join. For each resource type, there is a single customer class for the activity performed by the resources of the respective type.

2.1.2 Analysis of Queueing Networks

When a queueing network is fully specified according to Definition 1, one may employ numerous techniques from queueing theory to analyze performance based on the network (see [10] and the references therein). In the previous part, we define a very general formalism (Fork/Join networks, Definition 1), while most theoretical analyses in queueing theory assume much simpler models. Exceptions can be found in [6, 8]. In the absence of mathematical

tractability, the resulting models can be simulated to solve operational mining problems.

2.2 Event Logs

In this part, we define an event log (c.f. [66, Ch. 4]), which is the data model we use throughout this thesis. We use as an example a mapping of Dana-Farber Cancer Institute data into an event log.

2.2.1 Definition

In order to formally define an event log, we shall first define events and cases, then relate events to cases and define their attributes.

Definition 2 (Event, Case)**.** *Let $\mathcal{E}$ denote the set of all possible events, i.e. unique event identifiers. Let $\mathcal{E}^*$ be the set of all finite sequences over $\mathcal{E}$ with $\epsilon \in \mathcal{E}^*$ being the empty trace. The set of cases is $\mathcal{C} \subseteq \mathcal{E}^*$, i.e. a set of finite sequences of events. We require that each event appears at most once in some case.*

A case $c = \langle c_1, \ldots, c_n \rangle \in \mathcal{C}$ is thus a finite sequence of events such that $c_i \in \mathcal{E}, i = 1, \ldots, n$, with n being the number of events per case.

Events are associated with attributes, e.g., *timestamps*, *activities*, *locations* and *resources*. We denote by $\mathcal{A}_\mathcal{E}$ the set of all event attribute spaces: $\mathcal{A}_\mathcal{E} = \{A_i | i \in I\}$, with A_i being the ith attribute space and I being the index set of event attributes.

Definition 3 (Event schema, Event Attribute Function (EAF))**.** *An event schema, σ, is defined as a set of event attribute functions $\alpha_i : \mathcal{E} \to A_i$, where α_i is the ith event attribute function (EAF).*

For example, if A_i is the space of activities, then for an event $e \in \mathcal{E}$, $\alpha_i(e)$ is the corresponding service activity.

Similarly, cases are also associated with attributes, such as *a unique case identifier* or *service entity* (e.g., customer, and resource). We denote by $\mathcal{B}_\mathcal{C}$ the set of all event attribute spaces $\mathcal{B}_\mathcal{C} = \{B_j | j \in J\}$, with B_j being the jth attribute space and J the case attribute indices.

Definition 4 (Case schema, Case Attribute Function (CAF))**.** *The path schema, β, is defined as a set of all case attribute functions (CAF) $\beta_j : \mathcal{C} \to B_j, j \in J$.*

Cases consist of events, with each event having its set of attributes. Therefore, we may also have case attributes that would be aggregations of event attributes. The definition of such

Table 2.1: Day-Hospital data - Sample from Dec. 3rd, 2013

Service Path	Type	Event	Activity	Location	Timestamp	Transaction
12358	Patient	17201	Infusion	705C	1386074035	Start
12358	Patient	944871	Infusion	705C	1386092464	End
30395	Infusion Nurse	569764	Monitor	705C	1386075465	Start
30395	Infusion Nurse	569765	Monitor	705C	1386075496	End
30395	Infusion Nurse	570998	Leave	Staff Lounge	1386075465	Start

aggregated path attributes remains out of the scope of the thesis. We are now ready to define the event log.

Definition 5 (Event Log)**.** *An event log is a triple* (C, σ, β)*, where*

- $C \subseteq \mathcal{C}$ *is the set of observed cases,*
- σ *is the event schema, and*
- β *is the path schema.*

In the Findings chapter, we shall present event logs that are special cases of Definition 5. Note that there are two index-sets implicit in Definition 5: I associated with σ and J associated with β.

Example. Here we provide an example of an event log, as it is encountered in real-world data. Specifically, Table 2.1 presents an event log that comes from the Dana-Farber Cancer Institute.[1] The information was recorded by a real-time locating system (RTLS) on December 3rd, 2013.

The observed events set C corresponds to the rows in the table. The case schema, β, comprises a single CAF that maps cases into their corresponding types (e.g. patient, infusion nurse). The event schema, σ, includes the following EAFs: unique identifier, activity, location, timestamp and transaction attribute functions. We now provide a brief description of cases that are observed in the table. Patient 12358 started an infusion procedure at time $12:33:00$ in room $705C$. The infusion end time for that customer was recorded at $17:41:04$ (note that time is depicted in a universal timestamping method that counts seconds since 1/1/1970). An infusion nurse 30395 monitored several patients in the same room, including patient number 12358. The last activity of the nurse during that day was recorded in the staff lounge, as he was leaving the hospital.

In the remainder of this chapter, we formalize our operational mining problems using the above definitions of queueing models and event logs.

[1] The datasets throughout this thesis come from the Technion Laboratory for Service Enterprise Engineering (SEELab): http://ie.technion.ac.il/Labs/Serveng.

2.3 Operational Mining Problems and their Solutions

Operational process mining aims at solving operational problems. Below, we define operational mining problems, and show how performance analysis (e.g., time prediction), and operational conformance checking can be viewed as special cases of these problems.

Let $\mathcal{L}$ be the universe of event logs, which includes operational information on the process in its event and case schemata, e.g., timestamps, resources, etc. Further, let $\mathcal{M}$ be the universe of process models (e.g., F/J networks). We denote by $\mathcal{Y}$ the universe of (random) performance measures that we wish to predict, align, or improve in the operational setting. Examples of performance measures include the remaining time of a running case or resource utilization. Note that $Y \in \mathcal{Y}$ are random measures, which are distributions of random variables (measurements). Further, we assume that they can be (at least indirectly) quantified based on $L \in \mathcal{L}$.

Definition 6 (Operational mining problem)**.** *An operational mining problem consists of the triplet* $\mathcal{T} = \langle L, \{Y \in \mathcal{Y}\}, M \rangle$ *where*

- $L \in \mathcal{L}$ *is an event log,*
- $\{Y \mid Y \in \mathcal{Y}\}$ *is a set of random measures (or measurements),*
- $M \in \mathcal{M}$ *is a process model.*

When a model (log) is absent, we write $M = \perp$ ($L = \perp$) for the empty model (empty log). An empty set of measures is denoted by $\varnothing$. A missing model can be interpreted as a lack of prior (expert) knowledge on the underlying process. A missing log means that the problem is solved without data (e.g., via Operations Research). An empty set of measurements may correspond to unsupervised analysis, e.g., model discovery and cluster analysis. The problem of comparing an existing model M to an event log L (i.e., a conformance problem) with respect to alignment measures in $\mathcal{Y}$ can be considered as an instantiation of an operational mining problem. An example for a conformance-oriented problem is comparison of actual durations of a scheduled process to its planned counterpart [64].

The solution to an operational mining problem is essentially the mining of measures $\{Y \in \mathcal{Y}\}$ based on the log L and the model M. Continuing the above example, $\mathcal{T} = \langle L, \{Y_1, Y_2\}, \perp \rangle$ is the problem of predicting, based on L, remaining times for running cases (Y_1) as well as quantifying resource utilization for all resources (Y_2).

The definition of $\mathcal{T}$ can be mapped into the prediction framework as considered in the field of machine learning [34]. For example, consider the problem of predicting a single response variable $Y \in \mathcal{Y}$ (e.g., the delay of an arriving customer) based on a k-sized feature vector $X = (X_1, \ldots, X_k)$ (X_i being random variables). Features can include the queue

length at the time of the arrival, the waiting time of the highest-priority customer in queue (i.e. the head of the line), etc. Let L be the event log that records N samples of X and Y, i.e., $L = \{(x_1^j, \ldots, x_k^j, y^j)\}_{j=1}^N$. Then, the prediction of Y can be written as $\mathcal{T} = \langle L, \{Y\}, \perp \rangle$ (assuming that no process model is provided). One possible solution is a linear regression model that predicts Y as follows:

$$\hat{Y} = \beta_0 + \beta_1 X_1 \ldots \beta_k X_k. \tag{2.1}$$

Having defined operational mining problem, we now proceed to present the idea behind queue mining, which is our approach for solving congestion-related operational mining problems.

2.4 Queue Mining

Queue mining solves a given operational mining problem by discovering F/J networks from event logs and using the resulting network to solve the problem. Below, we first demonstrate our approach for F/J network discovery, and then present two approaches for solving problems: (1) *model-based queue mining*, and (2) *supervised queue mining*.

2.4.1 Discovery of Queueing Networks from Event Logs

Discovery of an F/J network from an event log involves the identification of the network structure (S), an estimation of the routing matrix (W), and a characterization of server dynamics (b), as follows.

Discovery of Structure and Estimation of Routing. To discover the general structure of a network, a large variety of existing process mining techniques can be used (see [66, Ch. 5] and the references therein). In particular, given that the schedule and the event log both contain resource types and start times of activities as well as their durations, we follow an approach that resembles the time-interval-based process discovery proposed by Burattin [15].

In [64] we employ interval algebra [4] and assume that resources of different types that perform activities concurrently at least once are indeed concurrent in the network. Subsequently, the required forks and joins are inserted, prior to resource nodes with more than one predecessor or successor, respectively. This yields a 100% fitting model, in the sense that all resource types and possible routing paths are represented. The step of fitting the structure may result in over-fitting.

For practical reasons, we also add input and output resource nodes with empty dynamics: an input node $s_i \in S_R$, to which all customers arrive (externally), and which is connected

to all servers with external arrivals; an output node $s_o \in S_R$ that is connected to all server nodes in which the process terminates according to the data.

To estimate the flow matrix W, we start by assuming that all weights are equal to 1. For edges that leave a fork or connect to a join, this weight remains unchanged. For all other edges between servers s and s' (s' can be a fork), the weights are set to Markovian probabilities, using the estimator that is calculated as the number of occurrences of the transition from s to s' in the data divided by the number of occurrences of s.

Lastly, weights for the input and output resource nodes are set as follows. The weight for a transition from s_i to s is set to the proportion of customers that arrive at node s out of all exogenous arrivals. Similarly, the weight for a transition from s to s_o is set to the proportion of services that terminate in s.

Characterizing Server Dynamics. The number of resources per node as a function of time, $\mathcal{R}_t$, is extracted from data using statistical methods as reported in [62]. The distribution of inter-arrival times $\mathcal{A}_t$ and processing times per class $\mathcal{B}_c$ can be fitted with the techniques presented in [13, 82]. Service policies, $\mathcal{P}$, can be discovered using the policy-mining techniques presented in [58]. The challenge of mining customer classes is addressed in [64].

2.4.2 Solving Operational Mining Problems with Queue Mining

Once an F/J network is fully specified, based on the event log, one may apply techniques that come from queueing theory for analyzing such networks, c.f. [10]. For example, one may use a queue-length type predictor to estimate the expected waiting time in a station [35, 62]. We refer to such queue mining techniques as *model-based queue mining*. A second approach for the analysis is by using the resulting F/J network in combination with historical data (i.e., the event log) for constructing an improved supervised learning framework [29]. We refer to the latter approaches as *supervised queue mining*. The two approaches clearly correspond to the ideas of *simulation mining*, and *supervised process mining*, respectively. Below, we briefly explain the two approaches based on a prediction operational problem with a single measure Y without a model in the background, i.e., the mining problem is $\mathcal{T} = \langle L, \{Y\}, \perp \rangle$.

Model-Based Queue Mining. Depending on the complexity of the resulting F/J network, we may apply different queueing analysis techniques ranging from simulation to tractable steady-state analysis [10]. If the model is a complex network of queues with concurrency and multiple customer classes, the analysis will rely mainly on stochastic simulation [30, 56], or approximation techniques [18, 79]. In contrast, for simple networks that

do not present time fluctuations, one may apply closed-form results from, at steady state, classical queueing theory [12, 36, 37].

Here we give an example of a discovered single-station single-class network with $G/M/s$ server dynamics, i.e., a general arrival process with i.id. inter-arrival times, i.id. exponentially distributed service times, and s independent homogeneous service providers. Further, the service policy is assumed First-Come First-Served; we propose methodologies for learning the policy from data, and testing whether an assumed policy holds [60, 63].

Based on the resulting model, we may apply a well-known predictor named the queue-length predictor (QLP) [78]. As its name suggests, it exploits the queue-length ahead of the customers whose delay time we wish to predict. Let Q be the current number of customers in a queue. The QLP for an arriving customer is:

$$\hat{Y}_{QLP}(Q) = \frac{(Q+1)}{s\mu}$$

with s being the number of service providers and μ being the service rate of an individual service provider. QLP is easily quantified from an event log that contains timestamps and event types (e.g. queue entry, service entry, and service completion).

The QLP was shown to work well with no abandonments. However, it fails when abandonments are prevalent (e.g., in call center scenarios) [62]. Further, queue-length predictors as well as other queueing theory techniques are mined, tested and extended based on real-world call center data in [63].

Supervised Queue Mining. In this part, we explain how the queueing perspective can be incorporated into (supervised) machine learning methods. The basis for these ideas can be found in the Findings chapter; specifically, the foundations are laid in [62], and [29]. Supervised queue mining currently comprises two approaches: (1) feature enrichment, and (2) model adaptation. Below, we provide some notation from the machine learning setting, and describe the two approaches in brief.

For notation, we return to the prediction problem, $\mathcal{T} = \langle L, \{Y\}, \perp \rangle$, with a single measure $Y \in \mathcal{Y}$ and an event log L (no existing model is assumed in the background). We assume that the event log contains N measurements of a P-featured vector X and measurement Y, $L = \{(x_i, y_i)\}_{i=1}^{N}$, with x_i being a single observation of the k-featured vector X (e.g., customer attributes, and queue length) and y_i being the corresponding observation of Y (e.g., an observed delay in a queueing station).

The supervised learning approach attempts to find an accurate model f that connects

X and Y up to some noise factor ϵ:

$$Y = f(X) + \epsilon. \tag{2.2}$$

The linear model that we mention in Eq. (2.1) is one example for f.

Approach 1: Feature Enrichment. In this approach, we define (feature) construction functions ϕ that transform the vector X into a new feature vector X_Q. These functions rely on the discovered queueing model, which contains information that reflects dependencies among running cases. The resulting features, X_Q, are assumed to better explain operational measures in $\mathcal{Y}$, i.e. the assumed framework changes to:

$$Y = f(X_Q) + \epsilon. \tag{2.3}$$

Note that with this approach the learning method f (e.g., a linear regression) is unchanged.

To illustrate the approach, consider a log L with the following event features: case identifier, activity, and timestamp. Applying a construction function ϕ that is based on the discovered F/J network to X can result in new queueing features, e.g., waiting time of the head-of-the-line, and current queue-length. Subsequently, these queueing features are plugged into the learning method f. This substitution was found to significantly improve prediction accuracy in [62] and [29]. Further, these additional features are often available in real-time and can be used to predict Y for feature values that were unobserved in L. For example, if one of the features in X_Q is the queue length Q and it reaches a level that was not observed in L, an adaptive algorithm may choose to predict Y based on the QLP. This gives the flexibility of using historical data vs. using a model-based approach [29], and in essence creates a hybrid approach that combines the best of both worlds.

Approach 2: Model Adaptation. An alternative approach to the aforementioned feature enrichment is an adaptation of the learning method f, based on the discovered queueing network. Here we use queueing predictors (e.g., the QLP) to adapt f. For example, in [29], f is a gradient tree boosting method that comprises K learners (decision trees), $f_1, \ldots, f_K$ [28]. Every learner is built on top of the other iteratively, i.e., $f_2 = g(f_1)$ with g being a boosting function. The resulting predictor f_K is used for the prediction. In [29], we construct the predictor f such that the first learner f_1 is based on another well-established result in queueing theory, which we refer to as the snapshot predictor [29, 62]. The other predictors (decision trees), $f_2, \ldots, f_K$, are constructed via boosting based on the initial queueing

predictor. In [29], we show empirically that the proposed method improves prediction accuracy both over the gradient tree boosting algorithm without the snapshot predictor, and over the snapshot predictor when it is applied without a learning procedure.

2.4.3 Multiple Queueing Models for Multiple Purposes

Queue mining is limited neither to the discovery of a single queueing model nor to the application of a single technique for solving one operational mining problem. On the contrary: given a set of problems, one is advised to discover different queueing models for the different types of analysis. Our studies show that, for time prediction, one would prefer a simple queueing model over complex simulation models. The former gives rise to accurate and efficient solutions. In contrast, for conformance checking, a complex queueing network is preferable, since it allows one to capture deviations in multiple dimensions, e.g., durations, resources, and customer classes. Therefore, queue mining can be viewed as a variety of solutions that can be combined to solve a set of operational mining problems.

Chapter 3

Findings

Queue Mining for Delay Prediction in Multi-Class Service Processes

Arik Senderovich, Matthias Weidlich, Avigdor Gal, Avishai Mandelbaum

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.

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.

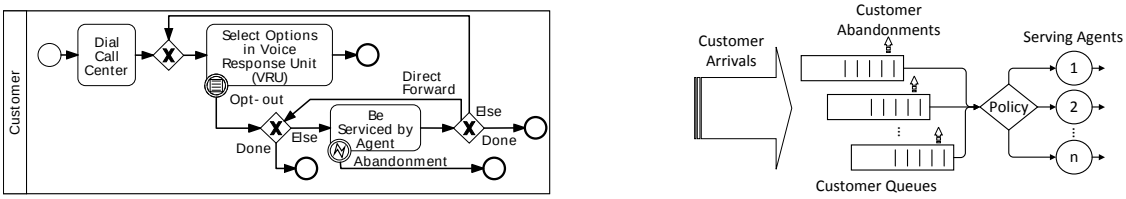
A specific operational problem that is affected by the competition of multiple cases for scarce resources is *online delay prediction*. This problem refers to the time that the execution of an activity for a particular case is delayed due to queueing effects. In a simple *single-class* setting, cases 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).

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



(a) Control flow perspective

(b) Queueing perspective

Figure 1: Example process in a call center

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 a call center agent (service provider). 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 *standardly* 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 resources the queueing station is denoted by $\mathcal{C}$, which stands for system capacity. When a case arrives and all service providers 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 a service provider 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. 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.

The queueing model. In this work, we focus on specific classes of queueing models. For the single-class queues, we shall assume the $G/M/s + M$ model and its variations. Specifically, this model assumes that arrivals come from a general distribution (e.g. Poisson process), service times are exponential and independent of the inter-arrival times, resource capacity is of size s, queue size is infinite, routing policy is FCFS and (im)patience is exponentially distributed.

This model is then directly lifted to the multi-class setting. For a multi-class system as the one depicted in Figure 1(b), factors that depend on the customer class are parametrized. In our case, one defines the arrival processes of each class to be Poisson processes (denoted M_i), and exponential service times (M_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 resources. 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 is typically FCFS, *within the same class*. For derivation of more accurate routing policies, see [10]. These underlying assumptions reflect upon our choices of relevant predictors and parameter estimation techniques throughout the paper.

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.

In practice, when incorporating a delay announcements mechanism into a recommender system, e.g. in call centers, banks and other service-driven organizations, the effect on *quality-of-service* can be both positive and negative. On the one hand, customers that hear the announcement might abandon and in turn cause lost business [30]. On the other hand, as mentioned above, announcements may relieve uncertainty and calibrate customer expectations regarding their remaining wait time. Another positive effect of abandonments caused by announcements is that customers that remain in the system wait much less. Since

customer patience and value are positively correlated, this phenomena has a positive effect on the firm's interest [29]. However, the topic of the operational influence of announcements on delays and abandonment is beyond the scope of the current paper. Here, we focus only on predicting the delays, with the goal of making the announced information as accurate as possible.

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 (or the root average squared 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:

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

3 Logs of Service Processes

The queue mining techniques developed in this paper exploit event logs recorded by an information system during the execution of a service process. The section gives an overview of the essential concepts related to these event logs.

An event recorded during the execution of a service process with a single-station queue captures the following information:

- the *time* of event occurrence;
- the *instance* of the service process (aka case), i.e., a specific customer with a request, who entered the system and has been served or is still being served;
- the *service transition*, i.e., the progress of the customer in the system;

- the *class* of the customer, which influences how the customer is served.

We denote the universe of all such events by $\mathcal{S}$, while $\mathcal{S}^*$ is the set of all finite sequences over $\mathcal{S}$. Formally, we model the information carried by an event as a set of function that assign attribute values to events. Here, time is modelled as UNIX timestamps; instance identifiers are natural numbers that refer to a specific customer with a request; service transitions refer to the arrival of a customer with a request in queue ($qArrive$), their abandonment ($qAbandon$), and the start ($sStart$) and end ($sEnd$) of the service that they receive; and the class is taken from a pre-defined categorical domain C of customer types, e.g., $C = \{VIP, Regular, LowPriority\}$.

Definition 1 (Event). *For the events $\mathcal{S}$ of a single-station service process:*

- $\tau : \mathcal{S} \to \mathbb{N}^+$ *assigns timestamps.*
- $\iota : \mathcal{S} \to \mathbb{N}^+$ *assigns instance identifiers.*
- $\epsilon : \mathcal{S} \to E = \{qArrive, qAbandon, sStart, sEnd\}$ *assigns service transitions.*
- $\xi : \mathcal{S} \to C$ *assigns classes from a set C of customer types.*

A set of example events are given in Table 1. Each of them describes the transition of a particular instance (for illustration purposes, we also indicated the name of the customer related to this instance) in the service process.

Table 1: Example events of a service process

Timestamp	Instance	Service Transition	Class
1415687360	1 (Daniel Will)	qArrive	VIP
1415687365	2 (John Smith)	qArrive	Regular
1415687366	1 (Daniel Will)	sStart	VIP
1415687381	1 (Daniel Will)	sEnd	VIP
1415687386	7 (Susan Lewis)	qArrive	Regular
1415687396	2 (John Smith)	sStart	Regular
1415687451	9 (Sarah Silver)	qArrive	LowPriority
1415688822	7 (Susan Lewis)	qAbandon	Regular

Following [3], we consider event logs that are defined as a set of *traces*. A trace is a finite sequence of events, all related to the instance of the service process and ordered by their time of occurrence. Since the events of an event log of a single-station service process have the specific structure outlined above, we refer to the respective log also as a *service log (S-Log)*.

Definition 2 (S-Log). *A service log (S-Log) $\Pi \subseteq \mathcal{S}^*$ is a set of traces, where for each trace $\langle s_1, \ldots, s_w \rangle \in \Pi$ it holds that $\iota(s_i) = \iota(s_j)$ and $\tau(s_i) \leqslant \tau(s_j)$ for $1 \leqslant i < j \leqslant w$.*

We observe that the events in Table 1 belong to four different traces, each comprising between one and three events.

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 traces 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 an event log, i.e., S-Log, as described above. From the S-Log, the set of possible service transitions is extracted, which is used to construct an initial transition system (TS). Then, the 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

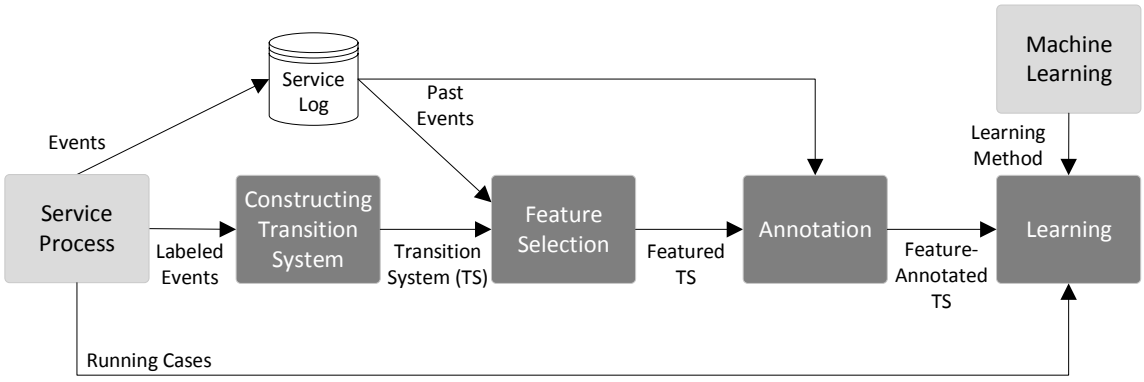


Figure 2: The featured transition system approach

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 function of the states and their features. Below, we formally define each state of our approach and demonstrate the three steps using the aforementioned example of a single-station service process.

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 service transitions that are defined by the investigated process and $T \subseteq \Sigma \times E \times \Sigma$ is the transition flow relation that describes state changes.

In single-station service processes, the set of service transitions is defined as $E = \{$qEntry, qAbandon, sStart, sEnd$\}$, cf., Section 3. Moreover, in this setting, the service transitions are also the basis for the definition of states. That is, a function π is defined that maps traces of an S-Log into some state abstraction. For our running example, we choose π such that it transforms traces into sequences of service transitions, i.e., $\pi : \Pi \rightarrow E^*$. As an example, let $p \in \Pi$ be the trace with instance identifier 1 in Table 1. Then, the result of operating function π on this trace is $\pi(p) = (qEntry, sStart, sEnd)$. However, other abstractions, such as the most recent service transition (memoryless), are also possible.

We are now ready to define the states of the transition system. The set of states Σ

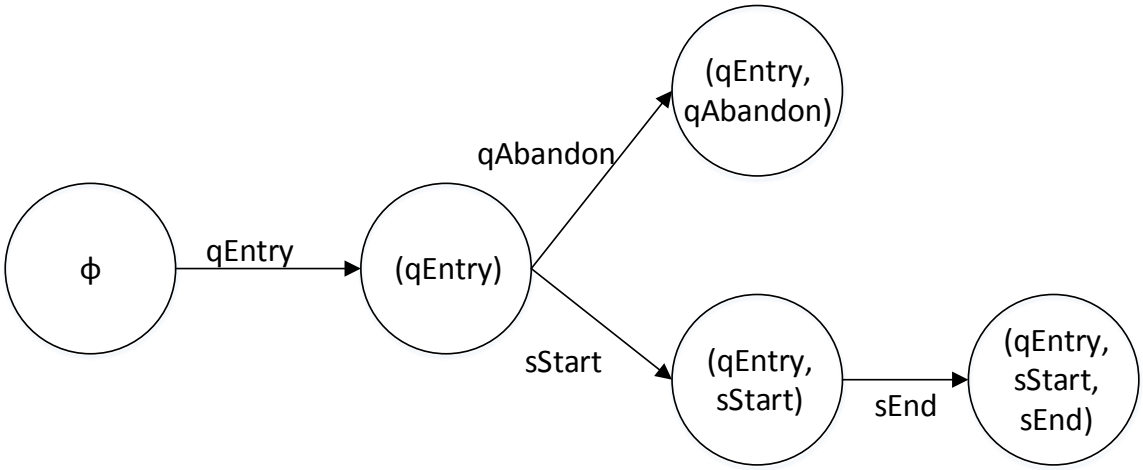


Figure 3: Single-station queue transition system

consists of abstracted traces that result from operating π on all prefixes of traces in Π:

$$\Sigma = \{\varnothing\} \cup \bigcup_{(s_1,\ldots,s_w)\in\Pi} \bigcup_{1\leqslant i\leqslant w} \{\pi((s_1,\ldots,s_i))\}$$

Customers that have not yet arrived into the system are in state $\varnothing$. According to this definition, the aforementioned trace of our running example ($\pi(p) = (qEntry, sStart, sEnd)$) induces four states, i.e., $\varnothing$, $(qEntry)$, $(qEntry, sStart)$, and $(qEntry, sStart, sEnd)$.

Finally, we define the last component T of the transition system. Let $\sigma.\sigma'$ denote the concatenation of two sequences of service transitions. Then, the state transitions T are defined as

$$T = \{(\sigma, e, \sigma') \in \Sigma \times E \times \Sigma \mid \sigma' = \sigma.(e)\}.$$

For our running example from Table 1, 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 the 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 Π. Each prefix of a trace Π is related to a single state in the transition system. From these prefixes, we mine the observed values x of the sequence of features X that are relevant for a particular state σ, i.e., $X = f(\sigma)$. This way, we get a sample of size N (given that there were N relevant events 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 instance 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 homogeneous. 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. $f(\sigma) = \varnothing, \forall \sigma \in \Sigma$. 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}:

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

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 s homogeneous working providers 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 service provider 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:

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

with s being the number of service providers and μ being the service rate of an individual provider.

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

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

Intuitively, when a target-customer arrives, it may progress in queue only if customers that are ahead of him enter service (when a resource 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. 5.2) reduces to the QLP predictor (5.2).

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 service providers s:

$$\begin{aligned} \widehat{q(t)} &= |\{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = qEntry \wedge \\ &\quad \wedge\ \tau(s_w) \leqslant t\}|, \\ \hat{s} &= |\{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = sStart\}|. \end{aligned}$$

In other words, the queue length is estimated by the number of paths that have experienced only a $qEntry$ event, while the number of providers 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 resources.

To obtain μ and θ we first define a auxiliary relations as follows:

$$\begin{aligned}
Q &= \{(p_1, p_2) \in \mathcal{S} \times \mathcal{S} \mid \exists\ (s_1, \ldots, s_w) \in \Pi, \\
&\quad i \in \mathbb{N}^+, 1 \leqslant i \leqslant w : s_i = p\ \wedge s_{i+1} = p'\} \\
R_1 &= \{(s_1, s_2) \in Q \mid \epsilon(s_1) = sStart \\
&\quad \wedge\ \epsilon(s_2) = sEnd\}; \\
R_2 &= \{(s_1, s_2) \in Q \mid \epsilon(s_1) = qEntry \\
&\quad \wedge\ (\epsilon(s_2) = sStart\ \vee\ \epsilon(s_2) = qAbandon)\}; \\
R_3 &= \{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = qAbandon\}.
\end{aligned}$$

Relation Q, indicates that two events follow each other directly in a trace. Relation R_1 contains pairs of events from the same trace that correspond to service start and end transitions, respectively. Similarly, R_2 contains pairs of events that are sequential in the same trace and indicate waiting in queue (until abandonment or service). Lastly, R_3 contains traces that ended with an abandonment. We use R_1 to estimate the average service time, m, as follows:

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

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,

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

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]. In other words, the system is assumed to be in a temporary steady-state, at least during the delay of the target customer.

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,

$$R_4 = \{s \in \mathcal{S} \mid \epsilon(s) = qEntry \ \wedge \ \exists (s_1, \ldots, s_w) \in \Pi : s_w = s\},$$

are the events of process instances that are currently waiting. The LES is estimated in two phases. First, we obtain the trace that has $sStart$ as the most recent event:

$$\upsilon = \text{argmax}_{(s_1,\ldots,s_w) \in R_5} \tau(s_w),$$

where,

$$R_5 = \{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = sStart\}.$$

Then, υ is the trace of the LES. Here, we assume that events are instantaneous and cannot co-occur. 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. That is, the state abstraction is now a function $\pi' : \Pi \rightarrow (E \times C)^*$ that maps traces of an S-Log into sequences that are built of pairs of a service transition and a class.

Considering the example trace $p \in \Pi$ with instance identifier 1 from Table 1, for instance, it holds $\pi'(p) = ((qEntry, VIP), (sStart, VIP), (sEnd, VIP))$. Deriving the states of the transition system with this adapted state transition means that the transition system for our example has no longer five states (cf., Figure 3), but 13 states (under the assumption that the customer class of an instance does not change during processing). For instance, the state $(qEntry)$ turns into three states $(qEntry, VIP), (qEntry, Regular), (qEntry, LowPriority)$.

Next, we adapt the used set of features. Instead of using the length of a single queue, a vector of queue-lengths is attached to each state. 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 customer classes.

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

$$\begin{aligned} \widehat{q_{c_i}(t)} \quad = \quad & |\{(s_1, \ldots, s_w) \in \Pi \mid \ \epsilon(s_w) = qEntry \\ & \wedge \ \tau(s_w) \leqslant t \ \ \wedge \ \ \xi(s_w) = c_i\}|. \end{aligned}$$

Once the vector of queue-lengths is mined, we apply the non-linear 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 $M_i/M_i/s + M_i$ 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 resources (or customers in service) $(r_1, ..., r_k)$ with $\sum_{i=1}^{k} r_i = r$ and r being the number of service providers online. We assume that the number of providers 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 $q_j(t)$ being the queue-length of the customer with *priority* j (not class j) at time t. 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 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 $(r^n_1, ..., r^n_k)$ to be the vector of customers in service (for each customer type) during the n-th iteration. For example, $r^1_1 = 3$, can be interpreted as 3 top-priority customers are being served at the first iteration of the algorithm. Let 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 non-negative).

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

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

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:

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

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 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:

$$\begin{aligned} r_1^{n+1} &= r_1^n + 1, \\ r_{\text{argmin}(n)}^{n+1} &= r_{\text{argmin}(n)}^n - 1, \end{aligned}$$

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

$$\psi^1_{QLPB} = \sum_{n=1}^{q_1+1} \frac{1}{(q_1 - n + 1)\theta_1 + \sum_{i=1}^{k} r_k^n \mu_k}.$$

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.

Remark 1. *Computationally, the iterative algorithm can be costly, since the number of iterations, per customer type is equal to the number of customers in both queue and service, which can become large. Therefore, an approximation to the quantity* $\sum_{i=1}^{k} r_k^n \mu_k$ *can be* $\sum_{i=1}^{k} r_k \mu_k$*, i.e. the number of service providers that serve each class (out of* r*) remains constant during the delay of the target customer. This approximation turned out to be accurate in our experimental setting.*

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 high-or-equal priority set, $H(c) = \{j|\eta(j) \leqslant \eta(c)\}$, as well as the vector of customers that are already in service, $(r_1, ..., r_k)$ (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

$$O^c \approx \sum_{i=1}^{\eta(c)-1} (\lambda_i \times W^c) \times \frac{1}{r\mu_i},$$

with W^c being the real waiting time of the target-customer (of class c), $\lambda_i \times W^c$ being the number of priority i (higher-priority) customers that arrived during the wait time of the target-customers and $\frac{1}{r\mu_i}$ is the approximated service time for these arriving customers.

For the n-th iteration, we assume that we have the service vector (number of active resources that serve the various customer types), given by $(r_1^n, ..., r_k^n)$ and as before we use the argmin(n) and 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 (1) and (1). Let $j = \eta(c)$ and denote $\rho_i = \lambda_i/(n_i\mu_i)$. Then we write:

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

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

$$\widehat{O^c} = \psi^j_{QLPU} \sum_{i=1}^{\eta(c)-1} \rho_i.$$

Therefore,

$$\psi^j_{QLPU} = \frac{\sum_{m=1}^{j-1} \psi^m_{QLPU} + \sum_{n=1}^{q_j+1} \frac{1}{(q_j-n+1)\theta_j + \sum_{i=1}^{k} r_k^n \mu_k}}{1 - \sum_{i=1}^{\eta(c)-1} \rho_i}.$$

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:

$$\psi^j_{QLPB} = \sum_{n=1}^{q_j+1} \frac{1}{(q_j - n + 1)\theta_j + \sum_{i=1}^{k} r_k^n \mu_k}.$$

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, r_j are trivial extensions of the mining methods proposed in the previous sections. To each of the methods, we add the predicate $\xi(s_1) = c$ in order to differentiate customer types. For example, given an S-Log Π, we estimate the number of top-priority customers in service as,

$$\widehat{r_1} = \left|\{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = sStart \wedge \xi(s_1) = 1\}\right|.$$

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 an S-Log Π:

$$\begin{aligned}\widehat{\lambda_j} \quad = \quad & |\{(s_1, \ldots, s_w) \in \Pi \mid \epsilon(s_w) = qEntry \\ & \wedge\ \xi(s_1) = c \ \wedge\ \tau(s_1) < t_2 \wedge\ \tau(s_1) \geqslant t_1\}|.\end{aligned}$$

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 an S-Log Π and time t, this vector can estimated from the log as:

$$\widehat{h_{c_i}(t)} = \min_{\substack{s' \in \{s \in \mathcal{S} \mid \epsilon(s) = qEntry \\ \wedge\ \xi(s) = c_i\ \wedge\ \exists\ (s_1, \ldots, s_w) \in \Pi : s_w = s\}}} t - \tau(s').$$

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

7 Evaluation

This section presents an empirical evaluation of the delay predictors, for both single-class and multi-class scenarios. For the single-class scenario we test our techniques on real-world data, while for the multi-class scenario, we use both real-world data and synthetic logs.

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.

A sensitivity analysis of our approach is conducted on the synthetic logs. The results for the synthetic logs show several interesting phenomena, some of which add validity to our techniques, while others show in which situations the presented predictors mediocre accuracy.

Below, we first describe two real-world service logs and three synthetic logs that we used for our experiments (Section 7.1). Then, we describe the experimental setup and

mention implementation details of our approach (Section 7.2). We report on the main results in Sections 7.3 and 7.4, for the single-class and multi-class real-world data settings, respectively. Then, we present the results of the sensitivity analysis for the multi-class scenario conducted with synthetic logs. Finally, we discuss the results in Section 7.6 and conclude with applicability and threats to the validity of our approach (Section 7.7).

7.1 Data Description

The real-world 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 for 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). The synthetic data that we later use for sensitivity analysis comes from a set of simulation runs, based on a multi-class service process. We shall first provide a brief overview of the two real-world datasets. Then, we describe the assumptions and the generation of the synthetic data logs.

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 analysing 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.

Below, we provide the result of a short analysis of the Bank's dataset[2]. Figure 4 demonstrates the number of customers in queue on January 2nd, 2011. One may notice that the queue-length changes over the day and that the heavy load hours are between 9:00 and 11:00 in the morning (over 32 customers in queue), moderate load hours are between 8:00

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

[2] The statistical analysis was performed by SEEStat, a software for statistical analysis of service systems that is accessible online at http://seeserver.iem.technion.ac.il/see-terminal/

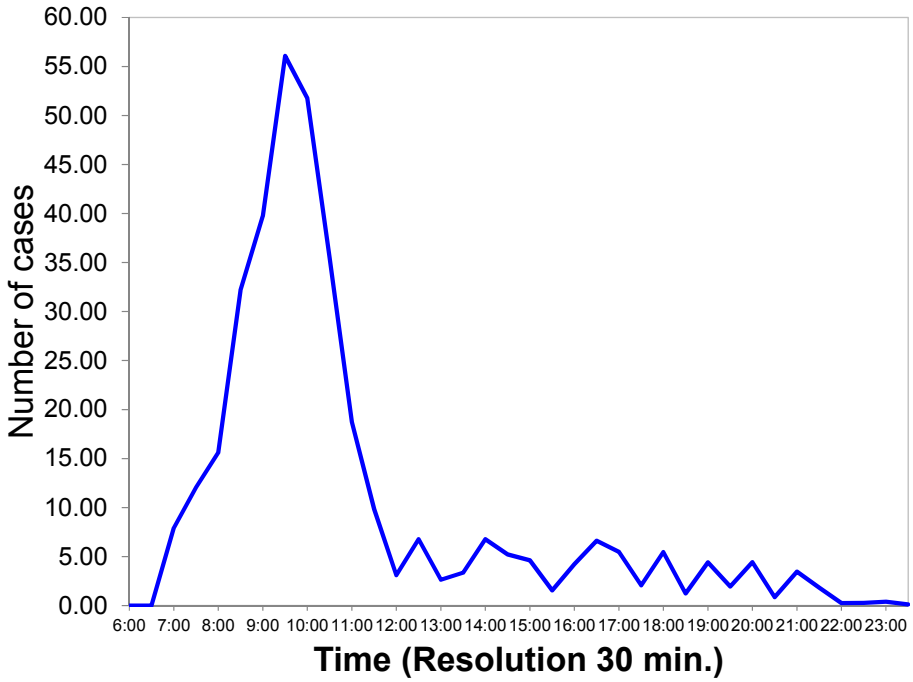


Figure 4: Queue-length as function of time-of-day (January 2nd, 2011)

and 12:00 excluding 9:00 to 11:00 (over 12 customers, less than 32 customers) and the rest would be considered typical load hours.

Figure 5 presents the mean service time over a single day (January 2nd, 2011, which is a typical working day in our training log). The horizontal axis presents the time-of-day in a 30 minutes resolution and the vertical axis presents the mean service time in seconds, during each of the 30 minutes. We see that the mean service time is mostly stable, and short on average, except towards boundaries (mornings and later afternoons).

Interestingly, the phenomena of slowdown that is well-known in service systems during busy hours is not observed. Between 9:00 and 11:00 in the morning, service times seem steady. However, an increase in service time (slowdown), would be an interesting 'what-if' analysis to conduct, when considering delay prediction. In case of service time increase, system load would also increase (assuming that arrival rates remain unchanged), and in turn, one would expect that snapshot predictors that are based on heavy-traffic approximations would be more accurate. On the other hand, a large increase in service times, with a decrease in the arrival rate should keep the system stable and thus keep the accuracy of the snapshot predictors unchanged across all runs. We will later explore these aspects with synthetic data in our sensitivity analysis.

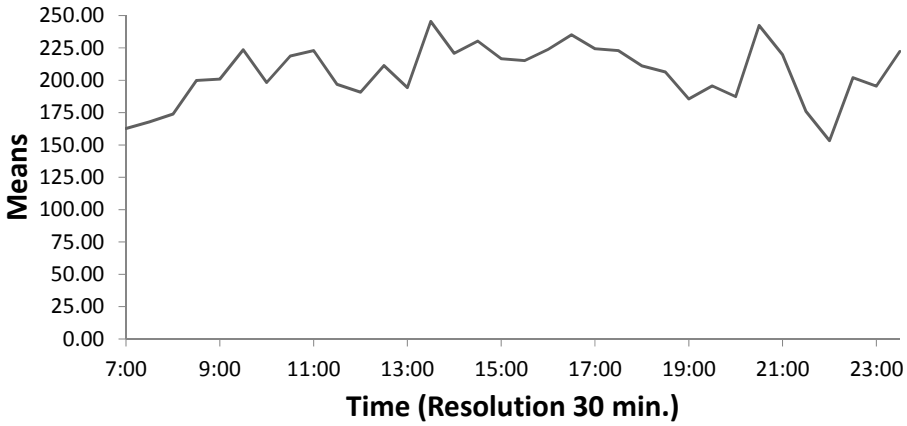


Figure 5: Service times as function of time-of-day (January 2nd, 2011)

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 resource positions on weekdays and 200-400 service providers 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.

In an exploratory data analysis of the data we observe similar behaviour compared to the first dataset. That is, load is time-varying, whereas service times are rather stable.

Synthetic Data of a Multi-Class Service System To create synthetic data for a sensitivity analysis, we simulated a $M_i/M_i/s + M_i$ system, which resembles the setting that we assume for the multi-class telecommunication company call center data. We modelled the system (with $i = 3$ classes) as a Coloured Petri Net (CPN) [33] and used CPNTools[3] to simulate different scenarios. As a *baseline* model, we inferred the parameters (arrival rates, service times, patience, number of servers) from the telecommunication company data, for a busy period (10:00-10:30). Then, we created three scenarios that correspond to three different 'what-if' analyses.

In the first scenario (Scenario 1), service times were gradually increased, while the rest of the parameters were kept constant. This increase imitates a 'disastrous' day in a service center, where workload of an already busy hour increases without additional resources.

The second scenario (Scenario 2) compares systems of various sizes. Specifically, we

[3] http://cpntools.org/

start with the baseline system and increase service times (by two-fold), while decreasing the arrival rates (also by two-fold). This created a stable queue-length and workload among the different runs with the emphasis being on the increasing durations of service. Here, patience was not changed, and thus even though services grew longer, customers were as (im)patient as in the baseline scenario.

Lastly, in the third scenario (Scenario 3) we repeat the procedure from Scenario 2, however we scale the patience to increase as service time durations become longer. This scenario is based on a well-known phenomena that experienced customers are willing to wait longer for long services [30].

For each scenario, we conducted several simulation runs while changing the respective parameters. For each run, we simulated 10 of 8 hours working days. Depending on the configuration, these runs featured between 83,000 and 660 process instances.

7.2 Experimental Setup

Controlled and Uncontrolled Variables The controlled variable in our experiments is the *prediction method* (or the delay predictor). The various methods that we defined in Sections 4 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 (see Equation 2 for the definition of RMSE). 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:

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

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.

For our synthetic experiments we have also considered as the uncontrolled variable the root mean relative error (RMRE), which can be written as:

$$RMRE = \frac{\sqrt{(\mathbb{E}[(W - \psi)^2]}}{\mathbb{E}[W]},$$

and approximated by the root average relative error (RARE):

$$RARE = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - p_i)^2}}{\frac{1}{N}\sum_{i=1}^{N} d_i^2}.$$

We add this performance measure to normalize the error with respect to the actual delays. This is due to the increase of service time in our experiments, that naturally leads to an increase in waiting times.

Implementation and Run-Time The prototypes for our algorithms that calculate the various predictors and their parameters were implemented in R[4]. Specifically, we used the implementations of the *rpart* package for decision trees, and the *mgcv* package for non-linear regression. The queueing algorithms were encoded manually, without reliance on existing packages. Data manipulations were performed in Visual Basic.

For learning-based methods, we divided the 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.

We finish this section with a brief discussion of the off-line and online computational effort that our algorithms require. We refer to run-time as the online cost, while off-line cost is assumed to be negligible (does not influence response time for online delay prediction). The run-time of the snapshot prediction algorithms is very fast, since it is assumed that the log contains all current queueing information.

For queueing models, the run-time of the iteration-based method (that approximates the upper and lower bounds) depends on queue-length and number of customers in service. For three months of data, it takes about 5 minutes (on an Intel Core i7, 16GB RAM computer) to run the iteration-based algorithm. However, in Section 6, we proposed an approximation that reduces the computational effort of the algorithm to several seconds. Another assumption is that queueing model parameters (arrival rates, service times, patience and number of expected resources) can be estimated off-line.

Transition-system techniques require intensive off-line learning (2 minutes run-time), but operate fast online, given a new target customer. However, if concept drifts occur during the day, which is a very likely scenario in uncertain systems it is expected that transition-based algorithms will in fact require online refinements and re-calculations.

[4] http://www.r-project.org/

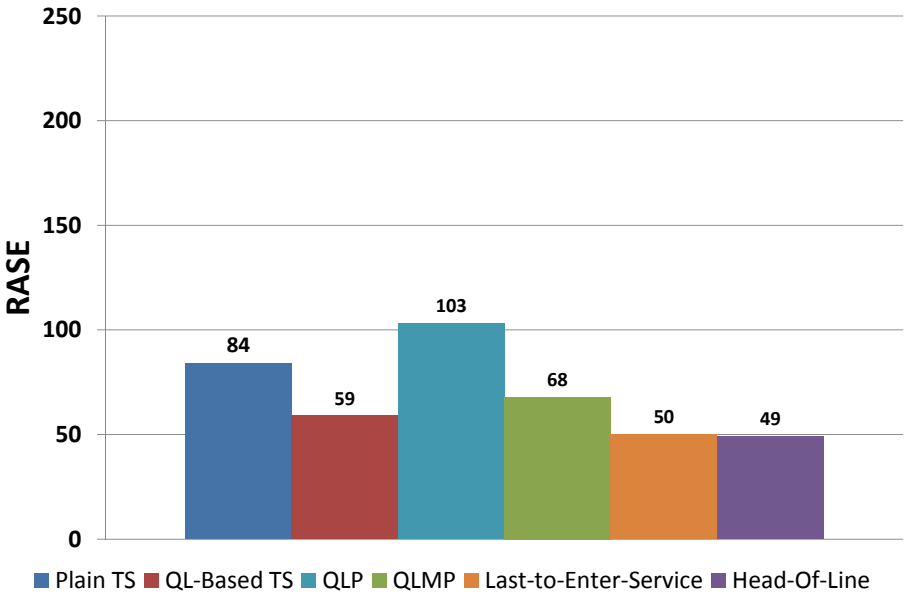


Figure 6: Prediction error for the first dataset: single-class techniques

7.3 Results: Single-Class Setting

Figure 6 summarizes the results that we received with the presented predictors for the single-class setting, i.e., the plain 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 behaviour 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 7 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 8 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 9 compares single-class and multi-

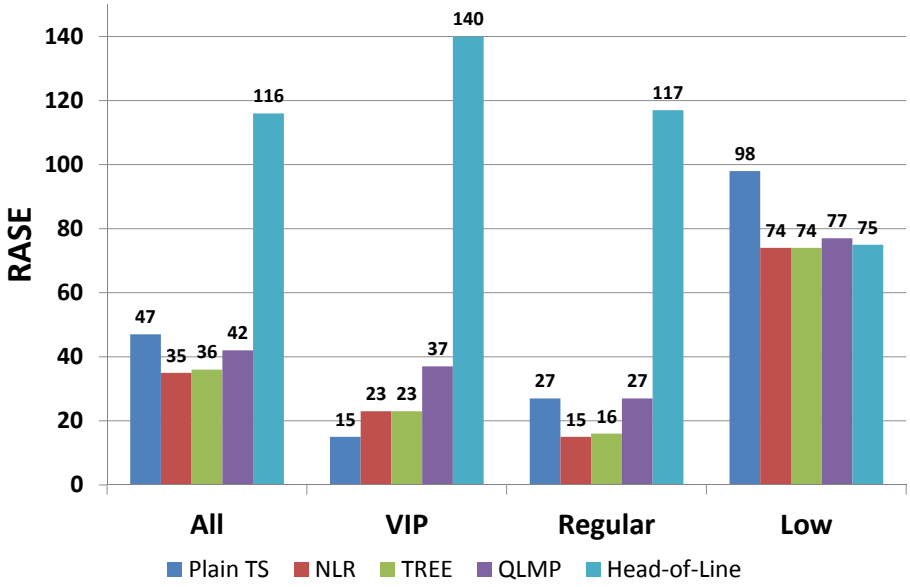


Figure 7: Prediction error for the multi-class dataset: single-class techniques

class methods for all customers and essentially merges the information that we gain from Figures 7 and 8. We observe a lower prediction error for all methods, when accounting for the existence of several customer classes.

7.5 Results: Multi-Class Setting in the Synthetic Log

Figures 10, 11 and 12 present the results of the three experimental settings of increasing load (Scenario 1), steady load (Scenario 2), and steady load with scaled patience (Scenario 3) in terms of absolute RASE values, respectively. In the first two scenarios, the horizontal axis, provides with the relative increase of service times with respect to baseline parameters: B being the baseline service time, which was estimated from a real-world log and $3B$ meaning that service time was increased by three-fold. For the third scenario, the horizontal axis represents the relative increase in both service times and customer patience. The vertical axis presents the RASE, with the different series in the chart corresponding to the various predictors.

For the experiments, we considered one of the predictors from each of the three classes. That is, the experiments include the predictor based on regression trees (Tree), the queueing model predictor (Queuing Model), and the Head-Of-Line (HOL) predictor. For the queueing model predictor, we found that there was no significance difference between the two bounds. Thus, the upper and lower bounds are a good approximation of the expected delay, under

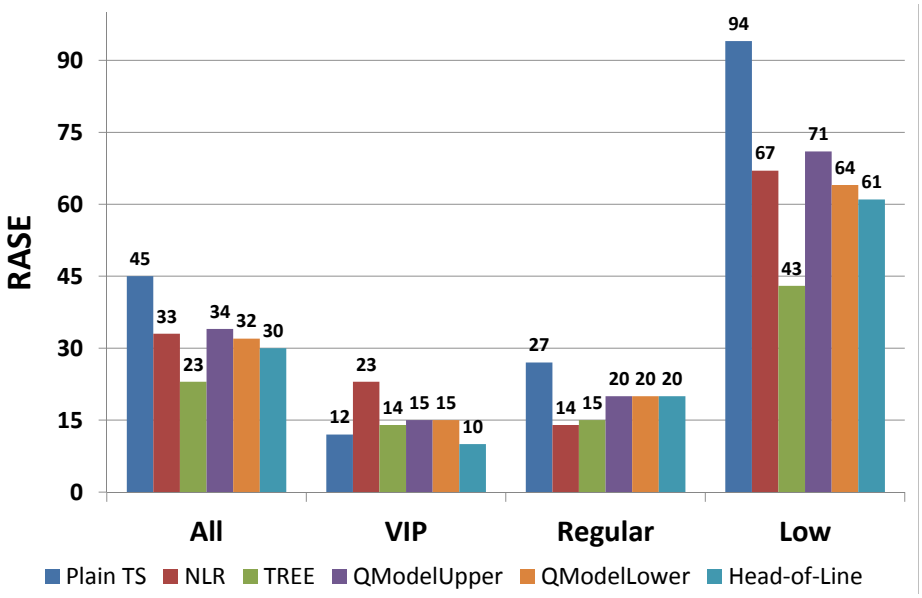


Figure 8: Prediction error for the multi-class dataset: multi-class techniques

model assumptions. This is consistent to the results of our previous experiments. Therefore, the figures always present an average of the upper and lower queueing bound, calculated with the approximation method presented in Section 6.2.

Figure 10 shows that when load increases, the Head-Of-Line predictor performs better than for an intermediate load, in which system is not in steady-state. Tree-based methods perform as well as the Head-Of-Line predictor, when load-increases. Since delays are more predictable in steady-state, less data is required for generalization. The queueing model predictors perform worse as load gets heavier. However, for lower values of the service time, it outperforms the snapshot predictor and there is a light improvement when service times are $8B$.

Scenario 2 (Figure 11) presents an interesting phenomena. For steady load with increasing service times, the snapshot predictor and the tree-based transition-system method perform adequately without a deterioration of their accuracy. The queueing model performs better than the other two predictors, as service times are close to baseline values. Then, its accuracy heavily deteriorates as service times increase and arrival rates decrease, and the predictor becomes incomparable to the other two predictors.

In Figure 12 (Scenario 3), we observe a similar phenomena among all predictor types. Here, all predictors fail to provide with an accurate prediction as service time and patience grow. However, when we observe the root mean relative error (measured by the root averaged relative error), depicted in Figure 13, we notice that the relative error is actually stable.

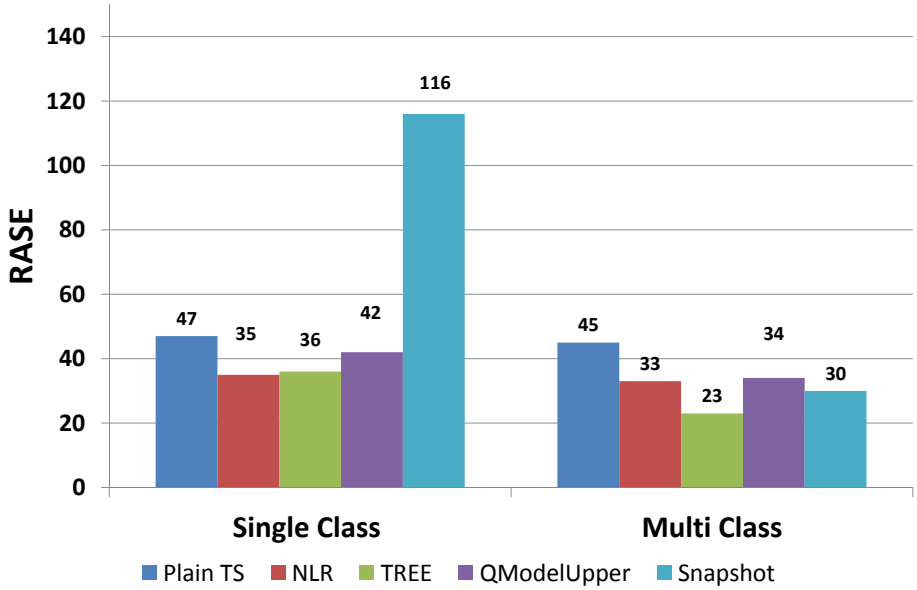


Figure 9: Prediction error for the multi-class dataset: single-class vs. multi-class techniques

7.6 Discussion

Below, we provide a three-part discussion of the results, with each part corresponding to the three sets of experiments (two real-world and synthetic data). In the first part, we discuss the difference in foundation between 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 analysing a multi-class system. Lastly, in the third part, we discuss the effect that system load has on the accuracy of prediction.

Data Mining Methods vs. Queueing Methods The single class analysis shows a slight superiority of the queueing methods (Figure 6). 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,

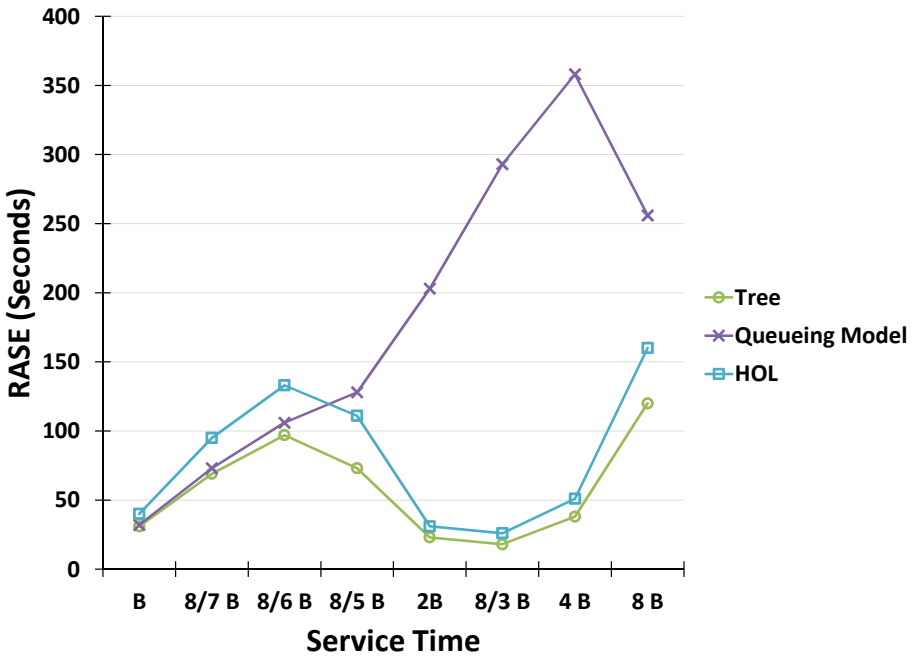


Figure 10: Scenario 1 – Root Average Squared Error for Increasing Workload

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.

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.

Analysing the classes separately can provide insights into the service process. For

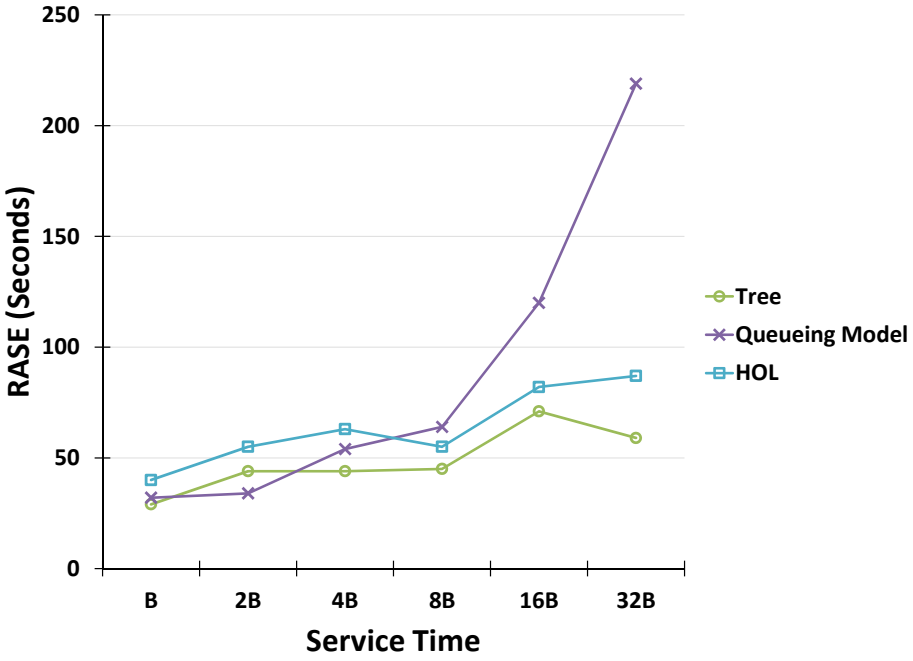


Figure 11: Scenario 2 – Root Average Squared Error for Steady Workload with Stable Patience

example, from Figure 8 we learn that the performance of the VIP system is stable, whereas low priority customers experience changes in load and thus in delay duration.

Effects of Load in Service Processes In this part, we mainly discuss the results of our synthetic experiments that we presented in 7.5. The term of system load is centric in service processes. The load represents the demand that arrives into the system, and is driven by the arrival rates, service times and customer patience. The relation between the demand for service (the load) and system capacity, dictates queue-length, delays, resource utilization and the probability that a customer abandons. In our sensitivity analysis with synthetic logs, we validated the queue mining methods against increasing load and stable load.

In the increasing load scenario (Scenario 1), when resource utilization increases and heavy-traffic conditions drive the system to a steady-state (of long queues and prolonged delays), the Head-Of-Line predictor performs very well. Specifically, Figure 10 shows a non-surprising result, that when a (heavy-traffic) steady state is reached (load increases), the Head-Of-Line predictor performs better than for an intermediate load. Tree-based methods perform as well as the Head-Of-Line predictor, when load-increases, since delays are more predictable in steady-state; less data is therefore required for generalization.

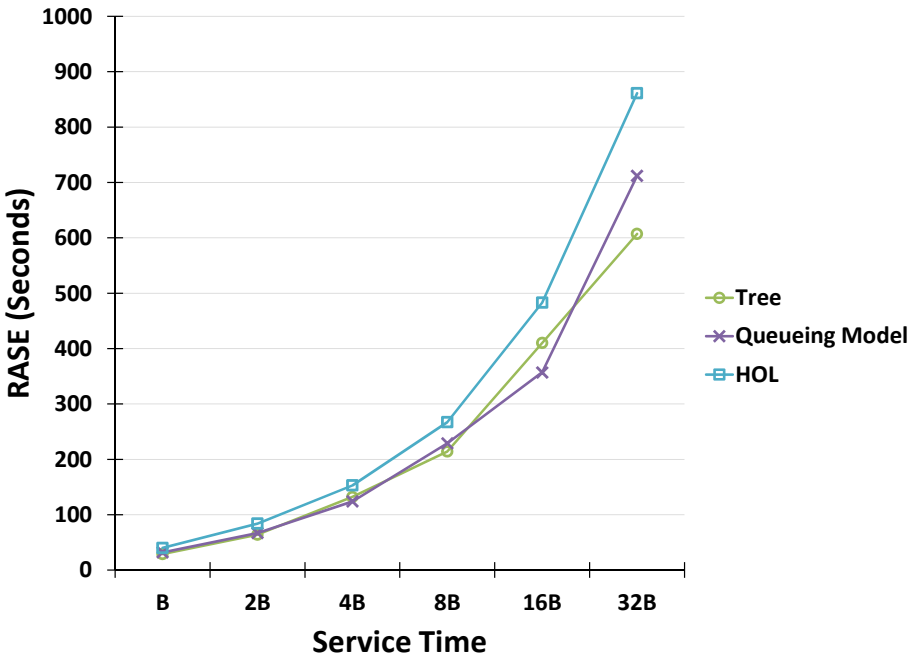


Figure 12: Scenario 3 – Root Average Squared Error for Steady Workload with Scaled Patience

In contrast, the queueing model seems to 'miss out' the steady-state that results from heavy-traffic conditions. The worsening of the queueing model predictor is related to its insensitivity to steady-state behaviour, being a time-varying (state-dependent) predictor. For intermediate load, the queueing model predictors are comparable to head-of-line and tree predictors and therefore can be considered accurate for delay prediction.

In the second scenario (Figure 11), when service times increase but load is stable, we observe that the snapshot predictor and the transition-based method are stable in performance. On the other hand, the queueing model predictor deteriorates. We explain this phenomena by the fact that in the second scenario we did not scale customer patience according to the increase in service times. A well-known phenomena in service systems is that customer patience is typically a function of the service time. Customers are willing to wait longer for a longer service [31]. Hence, we conducted the third experiment, and in Scenario 3 scaled the patience as well. Scenario 2 indicates that the Head-Of-Line and the transition-system methods are invariant to increase in service times. However, for the queueing model predictors, a refinement of experiments is required to check for their accuracy under increasing service times.

Scenario 3 is demonstrated in Figures 12 and 13; in this scenario the load is stable,

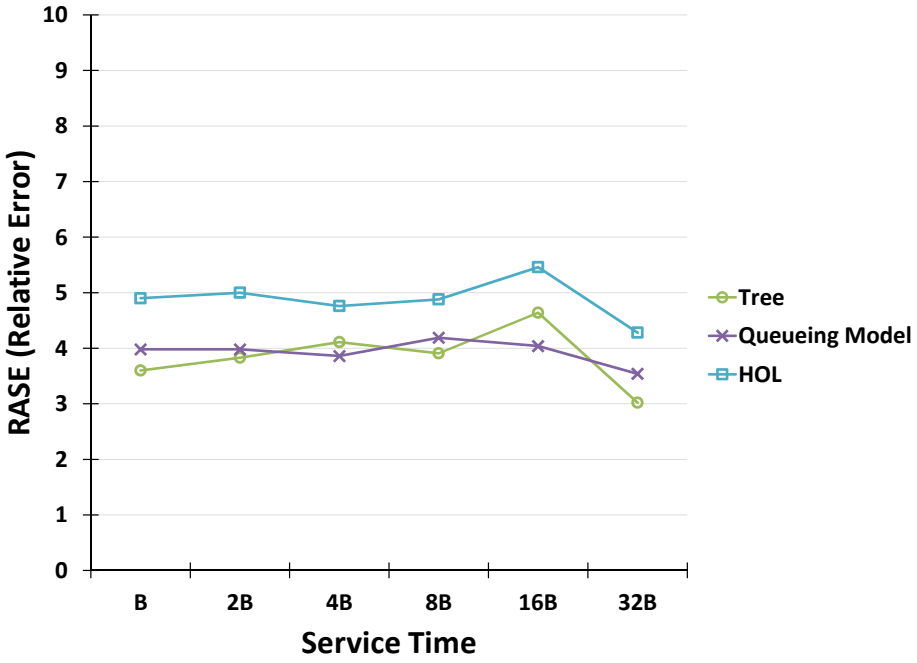


Figure 13: Scenario 3 – Root Average Relative Error for Steady Workload with Scaled Patience

although patience and service times increase by order of magnitude. For absolute values, all predictors perform worse as service times increase. However, when we consider the relative error, with respect to the average delay, we observe that all predictors are stable across simulation runs. This is an expected result, since for the same load and queue-lengths, the relative prediction accuracy per predictor should remain the same. Clearly, an error of 50 seconds is substantial for an average delay of 20 seconds and is considered low for average delay of 5000 seconds.

To summarize, service processes operate under varying loads. Some systems never experience heavy-load and therefore queueing model predictors can be adequate. For overloaded systems, snapshot predictors and learning methods seem to perform better when predicting delays. When load is stable, we expect that the performance of all predictors be adequate, regardless of the value of the load.

7.7 Applicability and Threats to Validity

Finally, we discuss the *applicability* and *threats to validity* of the predictors that are grounded in queueing theory. The transition-system predictor clearly suffers from state-space explosion, when applied to large logs with many possible routes (or activities). Otherwise, the method

is general enough to be applied for any process model with corresponding features.

Therefore, we divide the discussion into two parts, according to the two queueing predictor types. For every predictor-type, we discuss its applicability by over-viewing the set of assumptions under which these predictors can be applied. Then, we provide some empirical evidence that may constitute a threat to the validity of our approach.

Snapshot Principle Predictors: Heavy-Traffic Assumption Throughout our real-world data experiments, snapshot predictors outperformed the queueing model predictors in their precision. 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 either the current Head-Of-Line (HOL) or the delay of the Last-Customer-To-Enter-Service (LES). Our main insight is that in service processes one must often consider only recent delay history when inferring on newly arriving cases, since that customer arrives into a temporary 'steady-state'. This is especially characteristic of periods in which the system is in 'heavy-traffic', since heavy-traffic assumptions imply a steady-state. In 'heavy-traffic' the system changes negligibly during the processing time of the target customer. Moreover, the snapshot principle was shown to work well with multiple service stations as well [21]. Therefore, investigating its applicability to complex processes that can be represented by a network of queues may provide competent prediction.

The applicability of the snapshot predictors can be threatened by the absence of heavy-traffic conditions in the service process. If system state changes frequently, then even if the load is heavy, these conditions are unfulfilled. For instance, during certain hours of the day the service process can be in light-traffic, and since call centers are time-varying systems, the steady-state assumption will not hold.

Indeed, as we observed in Figure 4, not all hours can be considered heavy, according to the formal conditions. However, in systems, such as our call center, it is often times so that the convergence to heavy-traffic limit (steady-state) is fast, and thus the method is still applicable. Moreover, service times in our system are comparable to the inter-arrival times of the customers, and therefore assuming that a customer will experience a similar delay with the Head-Of-Line is plausible. Lastly, when considering the time-changing system in a piecewise manner, often times (as we saw in our real-world data experiments), a steady-state constitutes a plausible assumption.

Queueing Model Predictors: Model Assumptions Matter The queueing model predictors consider the time-varying behaviour of the system and attempt to quantify the system-state based on the number of delayed cases. In other words, unlike the snapshot

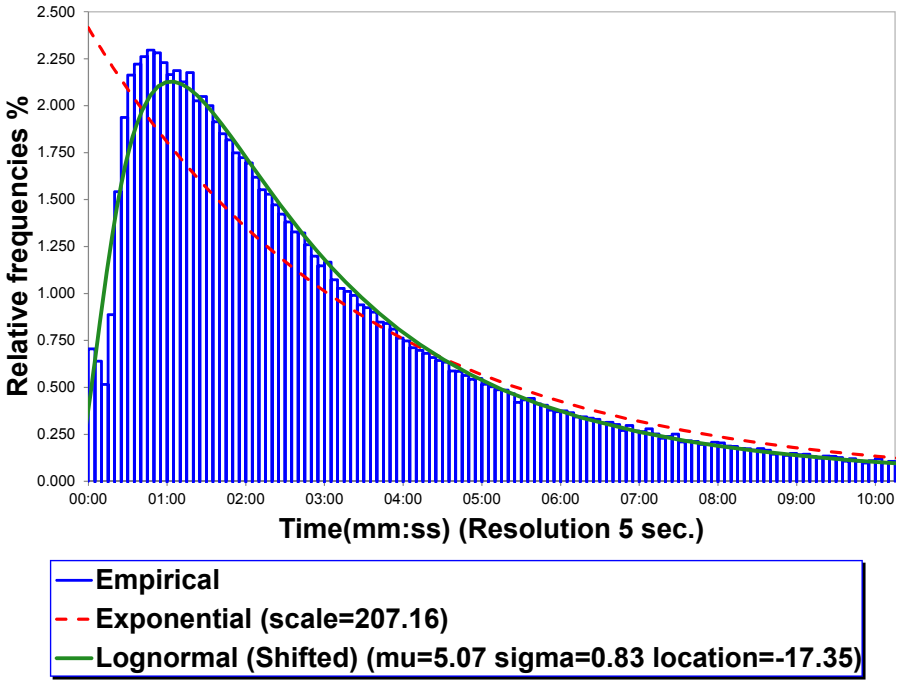


Figure 14: Service time distribution - data from January to March, 2011

predictors, no steady-state (while waiting) is assumed. The QLP method (for single-class queues) has mediocre accuracy, since it assumes that customers have infinite patience, which is seldom the case in real call center processes. We presume that the QLP would perform better for processes with negligible abandonment rates, e.g. in emergency departments and transportation.

On the other hand, the queueing models that do consider abandonments (QLMP in single-class and QModelUpper/Lower in multi-class), are comparable to the NLR transition-system method. Therefore, accounting for customer (im)patience is indeed relevant in the context of call centers, and other processes in which abandonments occur [23].

As we showed in our experiments, the queueing model predictors are often times inferior when compared to snapshot predictors or transition-system methods. This lack of accuracy can be explained by deviations between model assumptions and reality. To demonstrate this deviation, we further explore service times data from the Israeli bank's call center data.

Figure 14 presents the histogram of service times for all days in three months of the bank's call center data (January 2011-March 2011). The exponential distribution density is denoted by a dashed line, while the log-normal distribution is a solid line. We see a better fit of the empirical data to the log-normal distribution, which is a known phenomena in call centers [31].

However, it is often time noticed in the literature, that even when queueing model assumptions do not hold, the model has value when compared to data [32]. As such, we conclude that further investigation of the conditions under which the simplifying models hold in reality is required.

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 [20, 28]. 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 [18, 22]. 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%. Finally, we also reported on a sensitivity analysis of the predictors with synthetic data. By exploring scenarios of increasing load, various sizes, and under varying impatience. Here, methods based on transition systems that incorporate queueing information as well as snapshot predictors have been shown to be particularly robust across the different scenarios.

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. 1

[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. 1

[3] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011. 1, 7, 38

[4] W. M. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2) (2011) 450–475. 2, 3, 8, 12, 38

[5] R. W. Hall, Queueing Methods: For Services and Manufacturing, Prentice Hall, Englewood Cliffs NJ, 1991. 2

[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. 2

[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. 3, 21

[8] BPMN 2.0, Object Management Group: Needham, MA 2494 (2011) 34. 4

[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. 4

[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. 5, 22, 32

[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 38

[12] E. Nakibly, Predicting waiting times in telephone service systems, Master's thesis, Technion–Israel Institute of Technology (2002). 5, 39

[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. 5

[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). 5

[15] R. C. Larson, Perspectives on queues: Social justice and the psychology of queueing, Operations Research 35 (6) (1987) 895–905. 5

[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. 3, 8, 38, 39

[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. 12, 26

[18] R. Ibrahim, W. Whitt, Real-time delay estimation based on delay history, Manufacturing and Service Operations Management 11 (3) (2009) 397–415. 13, 14, 15, 39

[19] L. Schruben, R. Kulkarni, Some consequences of estimating parameters for the m/m/1 queue, Operations Research Letters 1 (2) (1982) 75 – 78. 14

[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. 14, 39

[21] W. Whitt, Stochastic-process limits: an introduction to stochastic-process limits and their application to queues, Springer, 2002. 14, 36

[22] W. Whitt, Predicting queueing delays, Management Science 45 (6) (1999) 870–888. 17, 39

[23] N. Gans, G. Koole, A. Mandelbaum, Telephone call centers: Tutorial, review, and research prospects, Manufacturing & Service Operations Management 5 (2) (2003) 79–141. 28, 37

[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. 38

[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. 38

[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. 38

[27] W. M. P. Van der Aalst, S. Dustdar, Process mining put into context, Internet Computing, IEEE 16 (1) (2012) 82–86. 38

[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. 39

[29] J. Huang, A. Mandelbaum, H. Zhang, J. Zhang, Refined models for efficiency-driven queues with applications to delay announcements and staffing, Tech. rep., Working paper, Technion (2014). 6

[30] A. Mandelbaum, S. Zeltyn, Data-stories about (im) patient customers in tele-queues, Queueing Systems 75 (2-4) (2013) pp. 115–146. 5, 25

[31] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, and L. Zhao. Statistical Analysis of a Telephone Call Center. Journal of the American Statistical Association, 100 (469) (2005) pp. 36–50.

[32] A. Mandelbaum, S. Zeltyn Service engineering in action: the Palm/Erlang-A queue, with applications to call centers, in: Advances in services innovations, Springer (2007) pp. 17–45. 34, 37

[33] K. Jensen, L. M. Kristensen, Coloured Petri Nets - Modelling and Validation of Concurrent Systems, Springer, 2009. 38
24

Traveling Time Prediction in Scheduled Transportation with Journey Segments

Avigdor Gal, Avishai Mandelbaum, Francois Schnitzler, Arik Senderovich, Matthias Weidlich

Abstract

Urban mobility impacts urban life to a great extent. To enhance urban mobility, much research was invested in traveling time prediction: given an origin and destination, provide a passenger with an accurate estimation of how long a journey lasts. In this work, we investigate a novel combination of methods from Queueing Theory and Machine Learning in the prediction process. We propose a prediction engine that, given a scheduled bus journey (route) and a 'source/destination' pair, provides an estimate for the traveling time, while considering both historical data and real-time streams of information that are transmitted by buses. We propose a model that uses natural segmentation of the data according to bus stops and a set of predictors, some use learning while others are learning-free, to compute traveling time. Our empirical evaluation, using bus data that comes from the bus network in the city of Dublin, demonstrates that the snapshot principle, taken from Queueing Theory, works well yet suffers from outliers. To overcome the outliers problem, we use Machine Learning techniques as a regulator that assists in identifying outliers and propose prediction based on historical data.

1 Introduction

Urban mobility impacts urban life to a great extent. People, living in cities, plan their daily schedule around anticipated traffic patterns. Some wake-up early to "beat" rush hour. Others stay at home and work during days when a convention comes to town. The pleasure of a night in the city may be hampered by the unexpected traffic jam in a theater vicinity and sometimes, even a minor fender bender at an urban highway may wrack havoc the schedule of numerous people.

To enhance urban mobility, much research was invested in traveling time prediction (c.f. [1] and the references within). That is, given an origin and destination, provide a passenger with an accurate estimation of how long a journey lasts. In particular, the ability to predict traveling time in scheduled transportation, *e.g.,* buses, was shown to be feasible [2, 3].

In this work, we investigate a novel use of methods from Queueing Theory and Machine Learning in the prediction process. We propose a prediction engine that, given a scheduled bus journey (route) and a 'source/destination' pair, provides an estimate for the traveling time, while considering both historical data and real-time streams of information that are transmitted by buses. To do so, we model buses as clients that go through a journey of segments that are interpreted as a network of queues. We propose a model that uses natural segmentation of the data according to intermediate stops. Using this model, common prediction methods that are either based solely on current snapshot data or that only exploit historic data are instantiated. We further present a novel set of predictors that combine the information gained from snapshot data with learning from historic data.

We test the proposed predictors using bus data that comes from the bus network in the city of Dublin. Our empirical analysis shows that the snapshot principle, taken from Queueing Theory works well yet suffers from outliers. To overcome the outliers problem, we use Machine Learning techniques that are based on historical data as boosting methods for the non-learning snapshot principle. To summarize, this work provides the following contributions:

- On a conceptual level, we propose a segmented model for traveling time prediction that enables exploitation of data about journey patterns in a fine-granular manner. We also show how this model is constructed from a log of recorded journeys.
- We outline how common prediction methods are instantiated for the traveling time problem and also develop a set of novel predictors that combine current snapshot data and learning from historic data.
- We present a comparative assessment of the developed prediction method using real-world data of the city of Dublin.

The rest of the paper is organized as follows. Section 2 develops the notion of a journey log to capture events of journeys. A definition of the addressed prediction problem is given in Section 3. Section 4 proposes the model of segmented journeys, followed by two prediction methods (Section 5). An empirical evaluation is given in Section 6. Section 7 discusses related work, followed by concluding remarks and future work (Section 8).

2 The Journey Log

Prediction of traveling time may exploit historical data on scheduled journeys or real-time streams of information on recent movements of vehicles. This section defines a common model for these types of information by means of the notion of a *journey log* (J-Log). A J-Log is a set of sequences of recorded journey events of scheduled bus trips, each sequence

being partially ordered by the timestamp that indicates the occurrence time of an event. A J-Log is a particular type of an event log, as they are known, for instance, in the field of business process mining, see [15, Ch. 4].

The presented notion of a J-Log refers to buses that emit *journey events*. Then, a journey is characterized as a sequence of journey events, which, for instance, signal that a particular bus reached a bus stop.

Definition 1 (Journey event, Journey)**.** *Let $\mathcal{E}$ denote a set of journey events. The set of all possible journeys is given as the set of finite sequences of journey events, denoted by $\mathcal{E}^*$.*

In the remainder, for a specific journey $j = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $n \in \mathbb{N}^+$, we overload set notation and denote by $e \in j$ an event e that is an element of the sequence $\langle e_1, \ldots, e_n \rangle$. We will also refer to the i-th event of a specific journey j by e_i^j.

Journey events are associated with attributes, e.g., *timestamps*, *journey patterns*, and *bus stops*. We model such an attribute as a function that assigns an attribute value to a journey event. A set of such attribute functions, in turn, defines the schema (aka structure or event type) over the set of journey events.

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

A journey log comprises *observed* journeys, such that each journey is formed of *observed* journey events emitted by a particular bus. Here, a function τ in the schema captures the timestamp of a journey event and the time in which the event $e \in \mathcal{E}$ occurred is denoted by $\tau(e)$. Further, journey events indicate the progress of a bus in its route; they represent the points in time that a bus reaches a specific bus stop. Such bus stops are modeled as a set $\mathcal{S} \subseteq \mathbb{N}^+$. Finally, journeys shall follow a predefined schedule, referred to as a *journey pattern*. It is modeled as a sequence of bus stops at which a bus should stop. Formally, a journey pattern belongs to the set of finite sequences over $\mathcal{S}$, which we denote by $\mathcal{S}^*$.

Generalizing the *functional* definition of an event log as presented in [16], a J-Log is defined as follows:

Definition 3 (J-Log)**.** *A journey log is a tuple (J, α_J), consisting of a set of journeys $J \subseteq \mathcal{E}^*$, which are defined by journey events of schema $\alpha_J = \{\tau, \xi, \pi\}$, such that*

- *$\tau : \mathcal{E} \to \mathbb{N}^+$ assigns timestamps to journey events,*
- *$\xi : \mathcal{E} \to \mathcal{S}$ assigns bus stops to journey events,*
- *$\pi : \mathcal{E} \to \mathcal{S}^*$ assigns journey patterns to journey events,*

Table 1: Notations for J-Log and the Online Traveling-Time Prediction Problem

Notation	Meaning
$\mathcal{E} \subseteq \mathbb{N}$	Set of all journey events
$\mathcal{E}^*$	Set of all sequences of journey events
$j \in \mathcal{E}^*$	A journey, i.e., a sequence of journey events
$\mathcal{S} \subseteq \mathbb{N}$	Set of bus stops
$\mathcal{S}^*$	Set of all journey patterns, i.e., sequences of bus stops
$\tau : \mathcal{E} \to \mathbb{N}^+$	Function assigning timestamps to journey events
$\xi : \mathcal{E} \to \mathcal{S}$	Function assigning bus stops to journey events
$\pi : \mathcal{E} \to \mathcal{S}^*$	Function assigning journey patterns to journey events
$T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$	Random variable for the traveling time from stop $\omega_1 \in \mathcal{S}$ to stop $\omega_n \in \mathcal{S}$ via the sequence of stops $\langle \omega_1, \ldots, \omega_n \rangle \in \mathcal{S}^*$, departing at time $t_{\omega_1} \in \mathbb{N}^+$

and it holds that $\tau(e_i) \leqslant \tau(e_{i'})$ *for all journeys* $\langle e_1, \ldots, e_n \rangle \in J$ *and* $1 \leqslant i < i' \leqslant n$.

An overview of the introduced notations, along with notations for concepts introduced later, can be found in Table 1.

Example 1. *We illustrate the notion of a J-Log using data of the bus network in the city of Dublin.*[1] *Here, location of buses is sampled in intervals of 5 to 300 seconds (20 seconds on average), depending on the current location of the bus. For each event the following data is submitted to a monitoring system:*

- *A timestamp of the event.*
- *A vehicle identifier for the bus.*
- *A bus stop relating the bus to the stop on its journey with maximal proximity. Hence, every event has a bus stop identifier, even when the bus is not at the stop.*
- *A journey pattern that defines the sequence of bus stops for a journey.*

Based on this input data, the construction of a J-Log as defined above is trivial; timestamps, bus stops and journey patterns are given directly in the input. To partition the events into journeys, a combination of the vehicle identifier and the journey pattern is used. An excerpt of the J-Log for the bus data from the city of Dublin is presented in Table 2. It features intuitive values for the attributes (e.g., stop "Parnell Square") as well as their numeric representation according to our formal model (e.g., 264 identifies the stop "Parnell Square").

[1] See also http://www.dublinked.ie/ and http://www.insight-ict.eu/

Table 2: Example J-Log from buses in Dublin

Event Id	Journey Id	Timestamp	Bus Stop	Journey Pattern
1	36006	1415687360	Leeson Street Lower (846)	046A0001
2	36012	1415687365	North Circular Road (813)	046A0001
3	36009	1415687366	Parnell Square (264)	046A0001
4	36006	1415687381	Leeson Street Lower (846)	046A0001
5	36009	1415687386	O'Connell St (6059)	046A0001
6	36012	1415687386	North Circular Road (814)	046A0001
7	36006	1415687401	Leeson Street Upper (847)	046A0001
8	36009	1415687406	O'Connell St (6059)	046A0001

3 Traveling Time Prediction: Problem and Approach

Traveling time prediction is an essential tool for providing integrated solutions for urban mobility, reaching from the creation of individual journey itineraries over capacity planning to car lift sharing [7]. In general, given a source and a destination, travel time prediction answers the question of how long the respective journey lasts. However, in most scenarios, traveling times vary greatly over time, *e.g.*, due to different levels of traffic load. Consequently, prediction is inherently time dependent, so that a specific predictor is a function of the source, the destination, and the time at which the prediction is made.

In this work, we address the problem of online travel time prediction in the context of a bus journey. That is, a journey may be ongoing in the sense that journey events already indicated the progress of the bus on its route. For such an ongoing journey, we are interested in the current prediction of the traveling time from the current bus stop to some destination via a particular sequence of stops, which is defined by the respective journey pattern. Using the model introduced in Section 2 for journeys and bus stops, we represent this *traveling time*, from stop $\omega_1 \in \mathcal{S}$ to stop $\omega_n \in \mathcal{S}$ via the sequence of stops $\langle \omega_1, \ldots, \omega_n \rangle \in \mathcal{S}^*$ and departing at time $t_{\omega_1} \in \mathbb{N}^*$ by a random variable $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.

The traveling time prediction problem relates to the identification of a precise predictor for $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.

Problem 1 (Online Traveling-Time Prediction)**.** *For a random variable* $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$ *representing a traveling time, the* online traveling-time prediction problem *is to find a precise predictor* θ *for* $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$.

Various measures that quantify the precision of prediction have been presented in the literature. In this work, we measure prediction precision by the root-mean squared error (RMSE), the mean absolute relative error (MARE), and the median absolute relative error (MdARE), to be defined in Section 6.1.

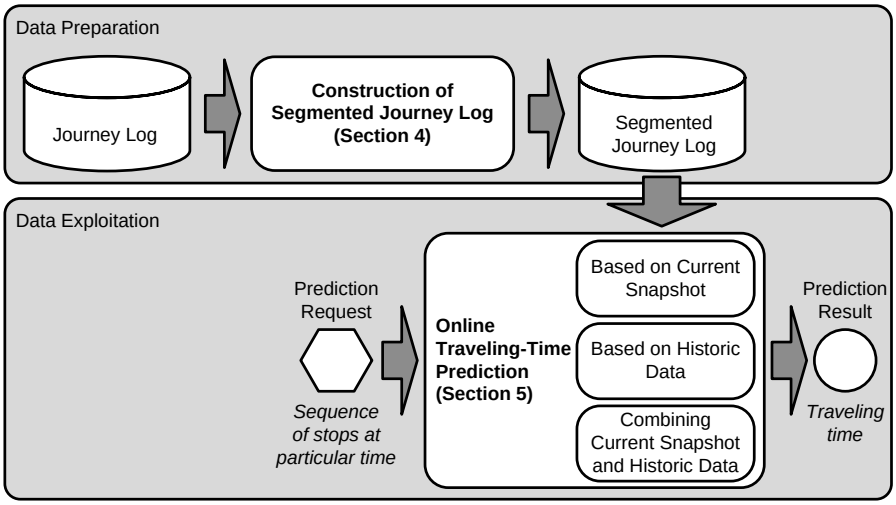


Figure 1: Our Approach to Traveling-Time Prediction

To solve the problem of online traveling-time prediction, we follow a two step approach. As illustrated in Figure 1, a first step prepares the input data in terms of a journey log by constructing a model that is based on the segments of journey patterns, referred to as segmented journey log. The second step exploits this model for the actual online traveling-time prediction. That is, given a prediction request that is characterised by a sequence of stops and a time for the departure, we rely on prediction methods to estimate the respective traveling-time. The prediction is based on the segmented journey log, but may use only the most recent information (i.e., the current snapshot), only historic data, or a combination thereof.

4 A Segmented Journey Model

In order to use data about journeys for traveling-time prediction, we construct a model that establishes a relationship between different journeys by means of visited bus stops. To this end, journeys are modeled as *segments of stops*.

A trip between two stops consists of segments, with each segment being represented by a 'start' stop and an 'end' stop, see Figure 2. Given the first stop of a trip $\omega_1 \in \mathcal{S}$ and the last stop of a trip $\omega_n \in \mathcal{S}$, the intermediate stops are known in advance since each bus follows a predefined journey pattern. Therefore, a trip can be described by segments that are characterized by a pair of stops of the form $\langle \omega_i, \omega_{i+1} \rangle$ (Figure 2). This segmented model, in turn, allows for fine-granular grounding of the prediction of traveling time $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$: instead of considering only journeys that follow the same sequence

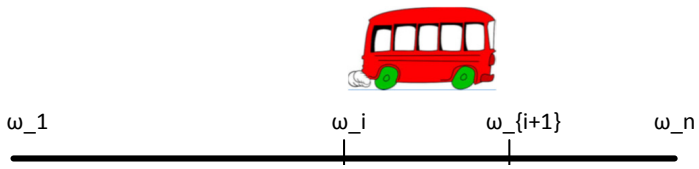


Figure 2: A segmented model of traveling times

of stops $\langle \omega_1, \ldots, \omega_n \rangle$, all journeys that share some segments can be used for prediction.

Below, we first describe the segmented model for journeys (Section 4.1). Subsequently, we show how to transform a J-Log into a Segmented J-Log, a log that contains information on segments (Section 4.2).

4.1 The Segment Model

While the benefit of segmentation of journeys for the purpose of time prediction has been widely acknowledged [8], various approaches may be applied to identify segments. Our work defines segments based on information about the bus stops of a journey pattern. Such segmentation is naturally derived from the structure of the data commonly available in traveling time prediction for bus networks. Moreover, such segmentation makes for effective prediction computation, where segments shared by more than one line can benefit from the frequent visitation of a segment by different bus lines. Our empirical analysis supports this claim. Finally, we note that such a segmentation is closely related to time prediction queries, where passengers identify their start and end stops thus allowing a straightforward translation of queries into our segment model. Such translation is the basis of our methods of learning and caching earlier computations for more efficient online computation.

Using information on bus stops, the prediction of the journey traveling time $T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1})$ is traced back to the sum of traveling times per segment. The traveling time per segment, in turn, is assumed to be independent of a specific journey pattern (and, thus, also independent of a specific journey):

$$T(\langle \omega_1, \ldots, \omega_n \rangle, t_{\omega_1}) = T(\langle \omega_1, \omega_2 \rangle, t_{\omega_1}) + \ldots + T(\langle \omega_{n-1}, \omega_n \rangle, t_{\omega_{n-1}})$$

where $t_{\omega_{n-1}} = t_{\omega_1} + T(\langle \omega_1, \omega_{n-1} \rangle, t_{\omega_1})$.

4.2 Deriving Segments from a Journey Log

A journey log (J-Log) is built of multiple journeys, where a journey is a sequences of events that are emitted by a bus as part of a particular trip. We rely on the aforementioned segment model as the basis for the traveling time predictors, and as a first step, we transform the J-Log into a Segmented J-Log that is built of timing information for segments.

A Segmented J-Log is a sequence of *segment events* that capture information on the start and end bus stop of the segment, the journey from which the segment event was derived, the respective journey pattern, and the start and end timestamps observed for the segment. The last two elements are computed using the earliest time the particular journey reached the start and end bus stop.

Definition 4 (Segment Events, Segmented J-Log)**.** *Let $\mathcal{G}$ be a set of segment events and let $\mathcal{G}^*$ be the set of all possible sequences of segment events. A Segmented J-Log is a tuple (G, α_G) where $G \in \mathcal{G}^*$ is a sequence of segment events of schema $\alpha_G = \{\xi_{start}, \xi_{end}, \epsilon, \pi_G, \tau_{start}, \tau_{end}\}$:*

- *$\xi_{start} : \mathcal{G} \to \mathcal{S}$ and $\xi_{end} : \mathcal{G} \to \mathcal{S}$ assign start and end stops to segment events,*
- *$\epsilon : \mathcal{G} \to \mathcal{E}^*$ assigns a journey to segment events,*
- *$\pi_G : \mathcal{G} \to \mathcal{S}^*$ assigns journey patterns to segment events,*
- *$\tau_{start} : \mathcal{G} \to \mathbb{N}^*$ and $\tau_{end} : \mathcal{G} \to \mathbb{N}^*$ assign start and end timestamps to segment events,*

A Segmented J-Log is constructed from the journey events of all journeys in a J-Log. That is, a segment event is derived from two journey events of the same journey, such that (1) the journey events refer to two successive bus stops of the journey pattern, and (2) the journey events are the earliest events referring to these two bus stops. A consists of the details of constructing the segmented J-Log.

5 Prediction Methods and Algorithms

The prediction methods we present can be divided into two categories. The first category does not generalize prediction from historical events, but rather uses recent events to predict future traveling times. As an example we present a method that comes from Queueing Theory and approximates systems in heavy-traffic. The second category is based on Machine Learning's decision trees. One feature of the feature space uses the method of the first category. Both prediction methods make use of the segmented model of journeys (Section 4.1) and the Segmented J-Log (Section 4.2).

5.1 Predicting Traveling Time using the Snapshot Principle

We now introduce the snapshot principle for traveling time prediction. Section 5.1.1 provides an overview of the method. The algorithm is detailed in Section 5.1.2.

5.1.1 Method

The *snapshot principle* [17], p. 187, is a heavy-traffic approximation that refers to the behavior of a queueing model under limits of its parameters, as the workload converges to capacity. In our context it means that a bus that passes through a segment, *e.g.*, $\langle \omega_i, \omega_{i+1} \rangle \in \mathcal{S} \times \mathcal{S}$, will experience the same traveling time as another bus that has just passed through that segment (not necessarily of the same type, line, *etc.*). Based on the above, we define a single-segment snapshot predictor, Last-Bus-to-Travel-Segment (LBTS), denoted by $\theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1})$.

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-world data sets. Results of synthetic simulation runs [18] show that snapshot-based predictors are indeed appropriate for predicting delays. Moreover, it was shown that the snapshot principle predict delays well when the system is not under heavy load and for a multi-class scenario [11, 12].

In our case, however, the LBTS predictor needs to be lifted to a network setting. In [19], it is stated that the snapshot principle holds for networks of queues, when the routing through this network is known in advance. Clearly, in scheduled transportation such as buses this is the case as the order of stops (and segments) is predefined. Therefore, we define a multi-segment (network) snapshot predictor that we refer to as the Last-Bus-to-Travel-Network or $\theta_{LBTN}(\langle \omega_1, ..., \omega_n \rangle, t_{\omega_1})$, given a sequence of stops (with ω_1 being the start stop and ω_n being the end stop). According to the snapshot principle in networks we get that:

$$\theta_{LBTN}(\langle \omega_1, ..., \omega_n \rangle, t_{\omega_1}) = \sum_{i=1}^{n} \theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1}).$$

We hypothesize that the snapshot predictor performs better whenever recent buses are in time proximity to the current journey. We test this hypothesis in Section 6.

5.1.2 Algorithm

We shall now demonstrate the algorithm to obtain the LBTS and thus the LBTN from the Segmented J-Log. Following the snapshot principle for networks, a bus that is currently at

ω_1 is to travel the sum of past traveling times of the last buses that traveled through each of the segments $\langle \omega_i, \omega_{i+1} \rangle$, prior to time t_{ω_1}. Therefore, for each pair $\langle \omega_i, \omega_{i+1} \rangle$ we are to search (in the Segmented J-Log) for the previous bus that passed through that segment (prior to time t_{ω_1}) and use its traveling time as an estimate for $T(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_i})$. Hence, the algorithm for extracting the snapshot predictor per segment (LBTS) is as follows. First, given a Segmented J-Log, (G, α_G), we obtain the segmented event that was last to travel through $\langle \omega_i, \omega_{i+1} \rangle$ prior to time t_{ω_1}:

$$s_i^{LB}(t_{\omega_1}) = \text{argmax}_{s \in G, \xi_{start}(s)=\omega_i, \xi_{end}(s)=\omega_{i+1}, \tau_{end}(s)<t_{\omega_1}} \tau_{end}(s). \quad (1)$$

Then, we calculate the estimator as:

$$\theta_{LBTS} = \tau_{end}(s_i^{LB}(t_{\omega_1})) - \tau_{start}(s_i^{LB}(t_{\omega_1})).$$

This definition characterizes θ_{LBTS} as a non-learning predictor—it only requires the traveling times of the last buses that went through particular segments.

5.2 Predicting Traveling Time using Regression Trees

In this section we describe the use of regression techniques to predict the bus traveling time $T(\langle \omega_1, \ldots \omega_n \rangle, t_{\omega_1})$. Unlike the snapshot method, regression trees exploit past journey logs to learn a prediction model, and then use this model to make a prediction on new instances of the problem, in our case, traveling times as part of current journeys.

Below, we first formalize the traveling times prediction problem as a regression problem and discuss the features we use. Then, we briefly describe the generic regression algorithms that we apply to solve the problem. Finally, we integrate the snapshot predictor with the regression algorithms.

5.2.1 Formalization and Features

Exploiting the introduced model of segmented journeys, we construct a predictor, θ_{ML},

$$\theta_{RM}(\langle \omega_i, \omega_{i+1} \rangle, t, \hat{t}_{\omega_i}) : S \times S \times \mathbb{N}^+ \times \mathbb{N}^+ \rightarrow \mathbb{R}^+ \ ,$$

for every traveled segment $\langle \omega_i, \omega_{i+1} \rangle$ along the route. The predictors takes as input the prediction time $t = t_{\omega_1}$ and the estimated time the bus enters the segment, $\hat{t}_{w_i}$. Based on

these predictors, a travel time $T(\langle \omega_1, \omega_n \rangle, t_{\omega_1})$ is estimated by

$$\theta_{RM}(\langle \omega_1, \omega_n \rangle, t_{\omega_1}) = \sum_{i=1}^{n-1} \theta_{ML}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1}, \hat{t}_{\omega_i}) \quad (2)$$

$$\hat{t}_{\omega_i} = t_{\omega_1} + \theta_{ML}(\langle \omega_1, \ldots \omega_i \rangle, t_{\omega_1}, t_{\omega_1}) \quad (3)$$

The predictor is formalized in two steps. First, we define a feature constructor $\phi : S \times S \times \mathbb{N}^+ \times \mathbb{N}^+ \rightarrow F$ where F is the feature space. The features are used as input for the second step, which is the construction of a regression model $\psi : F \rightarrow \mathbb{R}^+$ that outputs a traveling time prediction. The same ϕ is used for every segment while a model ψ is learned for each segment, anew. The features we consider are
(1) $\theta_{LBTS}(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_1})$, the travel time of the last bus that used that segment (see Eq. 1);
(2) $\hat{t}_{\omega_i} - \tau_{end}(s_i^{LB}(t_{\omega_1}))$, the interval between the time the last bus left the segment, and $\hat{t}_{\omega_i}$;
(3) $d(\hat{t}_{\omega_i})$, the day of the week; and
(4) $hms(\hat{t}_{\omega_i})$, the time of the day (hours, minutes, seconds) corresponding to $\hat{t}_{\omega_i}$.
The first two features are computed from the Segmented J-Log and therefore depend on the information available when the prediction is made, at time t.

5.2.2 Generic Algorithms

A regression tree [20] is a tree where each internal node is a test on the value of a feature, and where each leaf corresponds to a value of the target variable, in our case traveling time. An instance goes down from the root to a leaf by selecting at each internal node the branch corresponding to the result of the test of that node. The predicted value for that instance is the value associated with the leaf it reaches. In this section, we present the regression algorithms applied to the aforementioned features. These algorithms all output ensembles $\Psi_M = \{\psi_m\}_{m=1}^M$ of M regression trees. The value predicted by the ensemble is the weighted average of the values predicted by each tree of the ensemble:

$$\Psi_M(.) = \sum_{m=1}^{M} \lambda_m \psi_m(.) \ ,$$

where λ_m is the weight of tree ψ_m.

We selected ensembles of regression trees as the basis to our prediction methods due to their ability to automatically partition the feature space. Given categorical features, e.g. time-of-day, the resulting models will provide a clear breakdown into the different categories. Using an ensemble rather than a single model typically leads to an improvement

in accuracy [21]. We briefly describe below the methods we considered to build ensembles.

A **random forest** (RF) [22] is an ensemble built by learning each tree on a different bootstrap replica of the original learning set. A bootstrap replica is obtained by randomly drawing (with replacement) original samples and copying them into the replica. Each tree is learned by starting with a single leaf and greedily extending the tree. An extension consists of considering all possible tests (features and values) at all leafs and splitting the leaf using the test that maximizes the reduction in quadratic error. The tree weights are all equal $\lambda_m = 1/M$.

Extremely randomized trees (ET) [23] is an ensemble where each tree was learned by randomizing the test considered during greedy construction. Instead of considering all values of the features for the split test, only a value selected at random is considered for each feature (and leaf). The tree weights are all equal $\lambda_m = 1/M$.

AdaBoost (AB) [24] builds an ensemble iteratively by reweighting the learning samples based on how well their target variable is predicted by the current ensemble. The worse the prediction is, the higher the weight becomes. Therefore, the next tree constructed focuses on the most 'difficult' samples. Given the m^{th} model $\psi_m : \mathcal{X} \to \mathcal{Y}$ learned from a learning set $\{x_k, y_k\}_{k=1}^N$ with weights w_k^m, the next weights w_k^{m+1} are computed as follows:

$$
\begin{aligned}
L_k &= (y_k - \psi_m(x_k))^2 / \max_j (y_j - \psi_m(x_j))^2 \\
\bar{L} &= \sum_k L_k w_k^m / \sum_j w_j^m \\
\beta_m &= \bar{L}/(1-\bar{L}) \\
w_k^{m+1} &= w_k^m \beta_m^{1-L_k} \quad .
\end{aligned}
$$

The value predicted is the weighted median of the predictions of the trees, where the weight of each tree ψ_m is $-\log \beta_m$. Initial weights are all equal to $1/N$. AdaBoost is typically used with weak models, which do not model the data well outside an ensemble. For this reason, the depth (number of tests before reaching a leaf) of regression trees is typically limited when they are combined using AdaBoost. In our experiments, we tried both a depth of 1 and 3. The latter was almost always better, so we will only report the corresponding results. Except for this limitation, trees are learned greedily on the re-weighted learning sets.

Gradient tree boosting (GB) [25] is another boosting algorithm. Instead of weighting the samples, GB modifies the target variable value for learning each tree. The values used

to learn the m^{th} tree are given by

$$\tilde{y}_k = y_k - \Psi_{m-1}(x_k) \ . \tag{4}$$

The new tree is trained on the prediction error $\tilde{y}_k$. In this algorithm, the model weights are replaced by leaf weights λ_m^l, where l is a leaf index. The leaf weight is given by $\lambda_m = \nu\gamma_m^l$. ν is a regularization term (equal to 0.1 in our experiments). γ_m^l is optimized by line search:

$$\gamma_m^l = \arg\min_{\gamma} \sum_{k:reach(x_k,\psi_m,l)} (y_k - [\Psi_{m-1}(x_k) + \gamma\psi_m(x_k)])^2$$

where $reach(x_k, \psi_m, l)$ is true if x_k reaches leaf l in the tree ψ_m. This ensemble is initialized by a tree learned on the unweighted learning set. We also considered a more robust version of this algorithm, denoted GBLAD, which optimizes the absolute deviation error instead of the mean quadratic error. The most important changes are that each tree is constructed on a learning set $\{x_k, sign(y_k)\}$, and the value of each leaf is the median of the prediction errors of the training samples that reach it.

5.2.3 Combining Snapshot and Regression Trees

The snapshot method stems from Queueing Theory and was demonstrated to perform well in practice, for delay prediction in various settings where heavy traffic (resource queues) produces these delays [11, 12]. Since bus delays are often induced by car traffic, it is tempting to use it as a baseline and try to improve over it. Boosting algorithms appear particularly suited for that task, since they construct ensembles of models sequentially, based on the results of the previous models in the ensemble. Following this line of reasoning, we modify the three boosting algorithms discussed above (AB, GB and GBLAD) to use the snapshot model as the initial model. We respectively denote the three resulting algorithms S+AB, S+GB and S+GBLAD.

6 Evaluation

In this section, we empirically evaluate the methods we proposed in Section 5. The main results of our experiments are:

- Prediction methods that combine the snapshot principle and regression tree techniques are superior, in terms of prediction quality, to performing separately either snapshot predictors or regression tress methods.

- Prediction error increases with the number of bus stops per journey. However, relative error is stable over trip lengths. As a result, prediction does not deteriorate proportionally to length of the journey (in stops).
- Somewhat surprisingly, the snapshot predictor performance does not deteriorate for longer trips, therefore contradicting the hypothesis that the snapshot predictor would be more precise for journeys with higher temporal proximity to the current journey.
- Prediction accuracy is negatively correlated with the number of buses traveling through the city (load proxy).

We first describe our experimental setup (Section 6.1), including controlled variables that were selected for measuring accuracy (prediction error). Then, we introduce the dataset used for our experiments (Section 6.2) by first going over the training set and then introducing the test set. Lastly, Section 6.3 reports on our main results.

6.1 Experimental Setup

For the regression trees methods, we used the scikit-learn [26] implementation to create ensembles of regression trees. Also, we relied on ensembles of $M = 100$ trees, unless otherwise stated. The algorithms that combine the snapshot method (S+AD, S+GB and S+GBLAD), contain 99 trees in addition to the snapshot model.

We consider several measures for the quality of the predictor θ. The Root Mean Squared Error (RMSE) measure is based on the squared difference between the real traveling time and the predicted value. Note that the measure can be calculated with respect to an entire trip, e.g. for θ_{LBTN} or for a single segment as in θ_{LBTS}. Formally, the RMSE is defined as follows:

$$RMSE(\theta) = \sqrt{\mathbb{E}[\theta - T]^2},$$

where $\mathbb{E}$ is the expectation of a random variable, T the real traveling time and θ the predictor for T. The RMSE quantifies the error in the time units of the original measurements, i.e., in our case trips are measured in seconds and therefore the error will also be returned in seconds. The theoretical measure of RMSE can be approximated from the *test set* as follows:

$$\widehat{RMSE}(\theta) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(t_k - p_k)^2},$$

where N is the number of trips (or segments), t_k the real travel times through the trip (or segment) and p_k the travel times predicted by θ for the corresponding trip.

The RMSE presents two major issues that require additional performance measures.

First, it is sensitive to outliers [27], and second it is not normalized with respect to the traveling time. To deal with the latter problem we consider the relative error, normalizing the error with respect to the real traveling time T. To accommodate for the first problem, we swap the squared error with the absolute error, which is known to be more robust to outliers [27]. Hence, we use the mean absolute relative error (MARE) and the median of absolute relative error (MdARE):

$$\begin{aligned} MARE(\theta) &= \mathbb{E}[\frac{|\theta - T|}{T}], \\ MdARE(\theta) &= \text{median}\left\{\frac{|\theta - T|}{T}\right\} \quad . \end{aligned} \tag{5}$$

For the empirical approximation of the MARE and the MdARE we use the following measures based on the test set:

$$\begin{aligned} \widehat{MARE}(\theta) &= \frac{1}{N}\sum_{k=1}^{N}\frac{|(t_k - p_k)|}{t_k}, \\ \widehat{MdARE}(\theta) &= \text{median}\left\{\frac{|(t_k - p_k)|}{t_k}, k = 1, \ldots, N\right\}, \end{aligned} \tag{6}$$

with N, t_k, p_k, defined as before and the median being calculated as the empirical 50th quantile.

6.2 Datasets

We shall now describe the construction of datasets used for training the regression trees and testing the proposed prediction methods. For a first set of experiments, we constructed training data using a single bus line, $046A$, which has a frequent and lengthy journey pattern. Then, the test set is given by a single day, for which we predict the traveling time for every possible pair of stations on the route.

A second set of experiments focuses on generality and stability of our prediction methods. Here, we constructed training data using four bus lines and test the prediction for single segments for a single day.

Training set The first training set consists of 8 days of bus data, between September and October of 2014. Each day contains approximately 11500 traveled segments. Essentially, the learning set is a Segmented Journey Log for those 8 days. Clearly, the first trip of each day does not have information of the last bus to go through a segment during that day. The data comes from all buses that share segments with line $046A$. The second training set

consists of 25 days worth of bus data, between September 1st and September 25th, for year 2014. The training set contains approximately *1,085,000* traveled segments (10 times more than in the first training set). Essentially, the learning set is a Segmented Journey Log for those 25 days. The data comes from four journey patterns (that correspond to bus lines) that are frequent and lengthy (046A0001, 046A1001, 400001, and 401001), and consists of all traveling times of bus lines that share segments with these journey patterns.

For each segment, we construct a discrete learning set:

$$LS(\langle \omega_i, \omega_{i+1} \rangle) = \{\phi(\langle \omega_i, \omega_{i+1} \rangle, t, t_{\omega_i}), T(\langle \omega_i, \omega_{i+1} \rangle, t_{\omega_i})\},$$

where $t_{\omega_i} \in \{\tau_{start}(s)\}_{s \in G_i}$ is the start time of a segment event and

$$G_i = \{s \in G : \xi_{start}(s) = \omega_i\} \tag{7}$$

is the set of segment events related to the segment $\langle \omega_i, \omega_{i+1} \rangle$. To construct this learning set, we should not only consider the case where $t = t_{\omega_i}$. Indeed, this corresponds to a prediction made when the bus enters the segment. With the exception of the first segment of the line, predictions can be required earlier than t_{ω_i} if the segment is not the first of the trip. A different prediction time may change the features, and it is important to have a learning set representative of the instances the predictor will process. Hence, we need to consider additional prediction times. Note that the features (see Section 5.2.1) only depend on t through $s_i^{LB}(t) \in G$, the segment event associated to the last bus that used $\langle \omega_i, \omega_{i+1} \rangle$ before t. Considering additional prediction times is therefore equivalent to considering the LBTS available at different prediction times.

There are multiple methods to generate such a learning set from a Segmented J-Log. We chose to estimate an upper bound $\Delta(\langle \omega_i, \omega_{i+1} \rangle)$ on the maximal interval between the time the last bus has left the segment and t:

$$\Delta(\langle \omega_i, \omega_{i+1} \rangle) = max_{s \in G}\big(\tau_{end}(s) - \tau_{start}(s_i^{LB}(t))\big) \ . \tag{8}$$

It is worth noting that although $\Delta(\langle \omega_i, \omega_{i+1} \rangle)$ is a bound on $\tau_{start}(s) - \tau_{end}(s_i^{LB}(t))$, we estimate it based on larger intervals, obtained by respectively replacing the two terms by $\tau_{end}(s)$ and $\tau_{start}(s_i^{LB}(t))$. We believe this increase will make a model learned on the resulting learning still relevant even if t_{ω_i} is overestimated.

We then build, for each $s \in G_i$, one learning instance for each segment event s' related

to the same segment and that finished less than $\Delta(\langle \omega_i, \omega_{i+1} \rangle)$ before s starts:

$$
\begin{aligned}
G_i^{LB}(s) = \Big\{ & s' \in G_i : \tau_{end}(s') \in \\
& \big[\tau_{start}(s) - \Delta(\langle \omega_i, \omega_{i+1} \rangle), \tau_{start}(s)\big] \Big\} \\
LS^s(\langle \omega_i, \omega_{i+1} \rangle) = \Big\{ & \phi\big(\langle \omega_i, \omega_{i+1} \rangle, \tau_{end}(s'), \tau_{start}(s)\big), \\
& T\big(\langle \omega_i, \omega_{i+1} \rangle, \tau_{start}(s)\big) \Big\}_{s' \in G_i^{LB}(s)} \\
LS(\langle \omega_i, \omega_{i+1} \rangle) = & \bigcup_{s \in G_i} LS^s(\langle \omega_i, \omega_{i+1} \rangle) \ .
\end{aligned}
$$

While this data set construction method introduces dependencies between learning instances, it also generates more instances to learn from.

Test set For the first part of the evaluation, the test set comprises of bus data from a single day, September 22nd, 2014. We considered actual trips of line $046A$, which is one of the lengthiest lines in the city of Dublin (58 stops). First, the line travels through city center, where traffic can become extremely hectic and a large number of passengers may cause stopping delays. Then, it goes through a highway section and lastly it visits a suburban area where the delays are mostly due to frequent get-offs of passengers [8]. During that day, the line has traveled through 111 journeys, all of which were included in our test set.

For example, a single journey of line $046A$ travels through $\langle \omega_1, ..., \omega_{58} \rangle$, and emits the following sequences: $\{\langle \omega_1, \omega_2 \rangle, \langle \omega_1, \omega_3 \rangle, \langle \omega_1, \omega_4 \rangle, ..., \langle \omega_2, \omega_3 \rangle, ...\}$. For every source-destination (e.g., $\langle \omega_1, \omega_3 \rangle$), the test set contains: (1) the source (ω_1) of the trip, (2) the destination (ω_3), (3) the traveling time between source and destination, (4) the time of entry to segment, (5) the number of segments between source and destination, and (6) a set of tuples that represent all segments between source and destination, including the entry time into each segment, and the duration of traveling through the segment.

Therefore, for our running example of trip $\langle \omega_1, \omega_3 \rangle$, our test set contains the following tuple:

$$
\begin{aligned}
\langle \omega_1, \omega_3, T(\langle \omega_1, \omega_3 \rangle, t_{\omega_1}), t, 2, & \\
& \{\langle \omega_1, T(\langle \omega_1, \omega_2 \rangle, \tau_{start}(s_1^{LB}(t_{\omega_1}))), \tau_{start}(s_1^{LB}(t_{\omega_1}))\rangle \\
& \quad \langle \omega_2, T(\langle \omega_2, \omega_3 \rangle, \tau_{start}(s_2^{LB}(t_{\omega_1}))), \tau_{start}(s_2^{LB}(t_{\omega_1}))\rangle\}\rangle,
\end{aligned}
$$

with the traveling time between the two stops, the entry time, the fact that the trip contains two segments, with ω_2 being the next stop after ω_1, the traveling time through and the entry

time into first $\langle \omega_1, \omega_2 \rangle$ and then $\langle \omega_2, \omega_3 \rangle$ of the last bus that went through each segment. For the second stage of our evaluation, we consider September 26th, 2014 as our test set. We considered all single-segment trips of four journey patterns, including of $046A$. These four patterns pass through Dublin city center, and are thus susceptible to time-of-day load variability.

6.3 Results

The results are divided into two experimental parts. The first considers whole trips with limited training set, and an extended test set, while the second builds upon an extensive training set, and a test set that consists of single-segment trips in the test set. This follows the paradigm of the segmented model we promote in this work, where improvement of single-segment prediction entails the entire model prediction improvement. We use the second part of the evaluation to show stability and generality by learning based on more data from four different bus lines. Moreover, we use the second part to test for time-of-day and load effects on the accuracy of prediction.

6.3.1 Part 1: Whole Trips

The first set of controlled variables that we present in the results are the *prediction methods*, including *snapshot predictor* (S), *random forest* (RF), *extremely randomized trees* (ET), etc. As additional covariate we consider the *length of trip*, that may vary between a single segment (length of 1) and a whole trip (e.g. for $046A$, 58 stops). Moreover, we performed an extensive analysis of the position of a segment in a trip.

Table 3 presents the accuracy of the different methods tested over all trips. In terms of root-mean square error, the snapshot method (S) is the least accurate, together with the random forest (RF) method. The square error of the combination of the snapshot principle and gradient tree boosting (S+GB) with respect to the square error of S is illustrated in Figure 3. There is a high density of trips whose S square prediction error is higher than the square prediction error of S+GB, showing once again that combining the snapshot method with random tree methods can lead to better predictions. In addition, it is interesting to note that the improvement is not restricted to large traveling times, which is likely to indicate the existence of outliers. As a side note, the vertical stripes visible for small prediction errors are due to the typical measurement acquisition frequency of 20 seconds. Indeed, the snapshot prediction typically results in an estimation error that is a multiple of 20, leading to these patterns.

Table 3: Accuracy of the prediction of the trip length, for the different methods tested over all trips

	S	RF	ET	AB	GB	GBLAD	S+AB	S+GB	S+GBLAD
RMSE	539	539	519	512	508	520	504	**494**	514
MARE (%)	23.37	24.11	22.05	27.08	20.46	19.38	26.32	19.95	**19.06**
MdARE (%)	16.15	16.37	15.23	18.05	13.84	13.86	16.84	**13.53**	13.65

Influence of Trip Length Figures 5 and 4 present the RMSE and MARE as function of trip length (number of stops on the route), respectively. For all methods, the RMSE increases as the trip length increases (from [83,96] to [1070,1296]). In contrast, the MARE decreases as trip-length increases, indicating that the error remains *proportionally* constant, regardless of the increase in trip-length. Moreover, methods that optimize the absolute error (GBLAD and S+GBLAD) perform better, as expected.

When observing Figure 4, we notice that the MARE of the snapshot predictor decreases at the same rate as the error of the rest of the methods. In other words, the last bus that traveled through a segment at the end of a long trip predicts the travel time as well as the last bus that traveled through an earlier segment, relatively to the accuracy of other methods. Again, this seems to contradict our hypothesis from Section 4. Lastly, we observe that random tree methods combined with the snapshot predictor generally improve over the same methods without the snapshot predictor.

Segment Position Figures 7 and 6 show the evolution of the root-mean square error and of the relative absolute prediction error (respectively) on the duration of a single segment as function of the order of segments in the trip. In other words, a trip that goes between stops ω_k and ω_l, with $l > k$, $index = 1$ is the segment $\langle \omega_k, \omega_{k+1} \rangle$. Therefore, these figures present the evolution of the incremental estimation error as the segment index increases.

According to Figure 6, the relative prediction error for all methods remain unchanged as the segment index increases, except for the latest segments where the value of the error grows significantly. We suspect that this occurs either due to the small test set size for trips of these lengths (only 111 samples for trips of 58 stops), or due to a deterioration of the prediction accuracy for lengthier trips. It can be observed that S+GBLAD, which is the combination between the snapshot predictor and the GBLAD learning algorithm, is the most accurate method *per segment*, as well as per trip (Figure 4).

Figure 7 presents surprising sharp decreases in the RMSE for some segment indexes

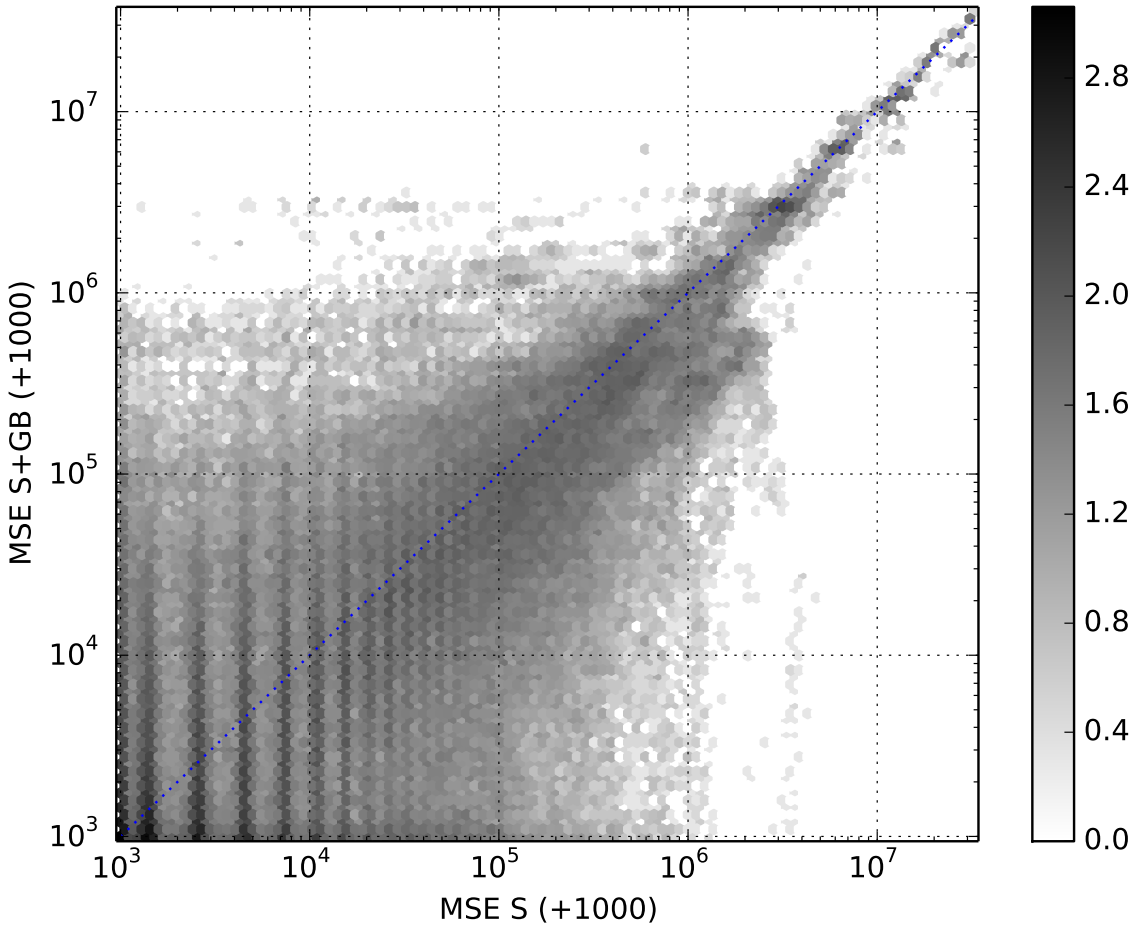


Figure 3: The density of test trips as a function of the square prediction error of S and S+GB shows that the prediction errors of S on many trips is above 10^6 while smaller than 10^6 for S+GB. All scales are logarithmic, and square errors have been increased by 1000 for plotting purposes.

(further segments have lower errors), with an increase towards the end. Intuitively, one may expect the accuracy of the snapshot principle, and maybe other regression methods, to decrease with the segment index (far segments). Indeed, segments further away in the trips have a larger time interval between the moment the journey we consider will enter the segment and the time the bus whose travel time is used as a prediction went through it.

This phenomena does not occur due to a difference between the nature of segments, but due to outliers, as can be seen in Figure 8. The figure contrasts box plots of the square estimation error (left) and the root mean square estimation error (right) for the snapshot

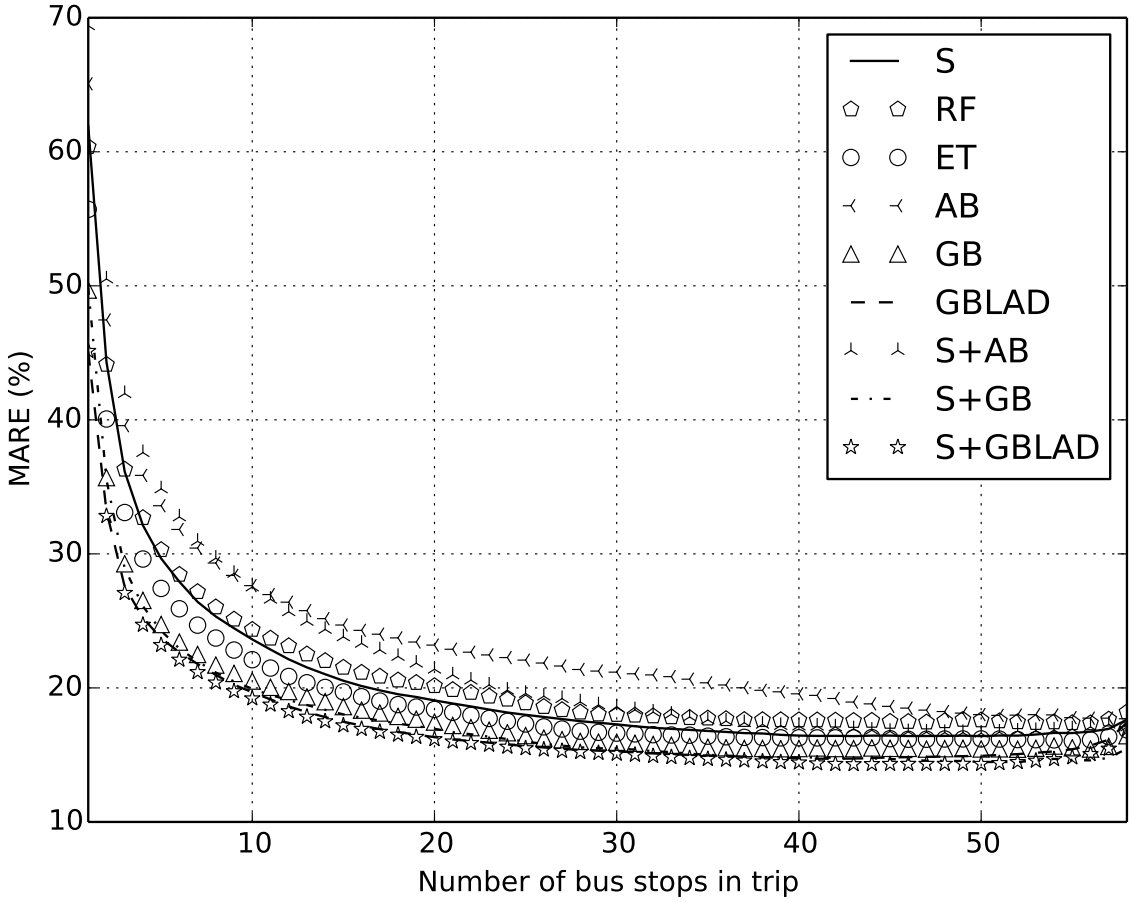


Figure 4: The mean absolute relative error on the duration of the whole trip decreases with the trip length. S+GBLAD is the best method for all trips length except for trips of length 1, where GBLAD is more accurate, and trips of length 52 and [55,58], where S+GB is more accurate.

method and for all segments at a given index in a trip. The drops in the curve of the RMSE correspond to the disappearance of outliers. Outliers are arranged in horizontal lines, because a single outlier in the journey log may affect several trips, i.e. an outlier in the segment $\langle \omega_i, \omega_{i+1} \rangle$ will affect the first segment of the trip whose source is ω_i, the second segment of the trip whose source is w_{i-1} etc.

Figure 7 does not allow analyzing the effect of the segment index on the precision of the snapshot method. However, Figure 8 contains the median of the square estimation error for this method, which is presented in the box-plots (at the lower part of the figure). One may observe that the median is stable with little variation in its values. Hence, the index of the

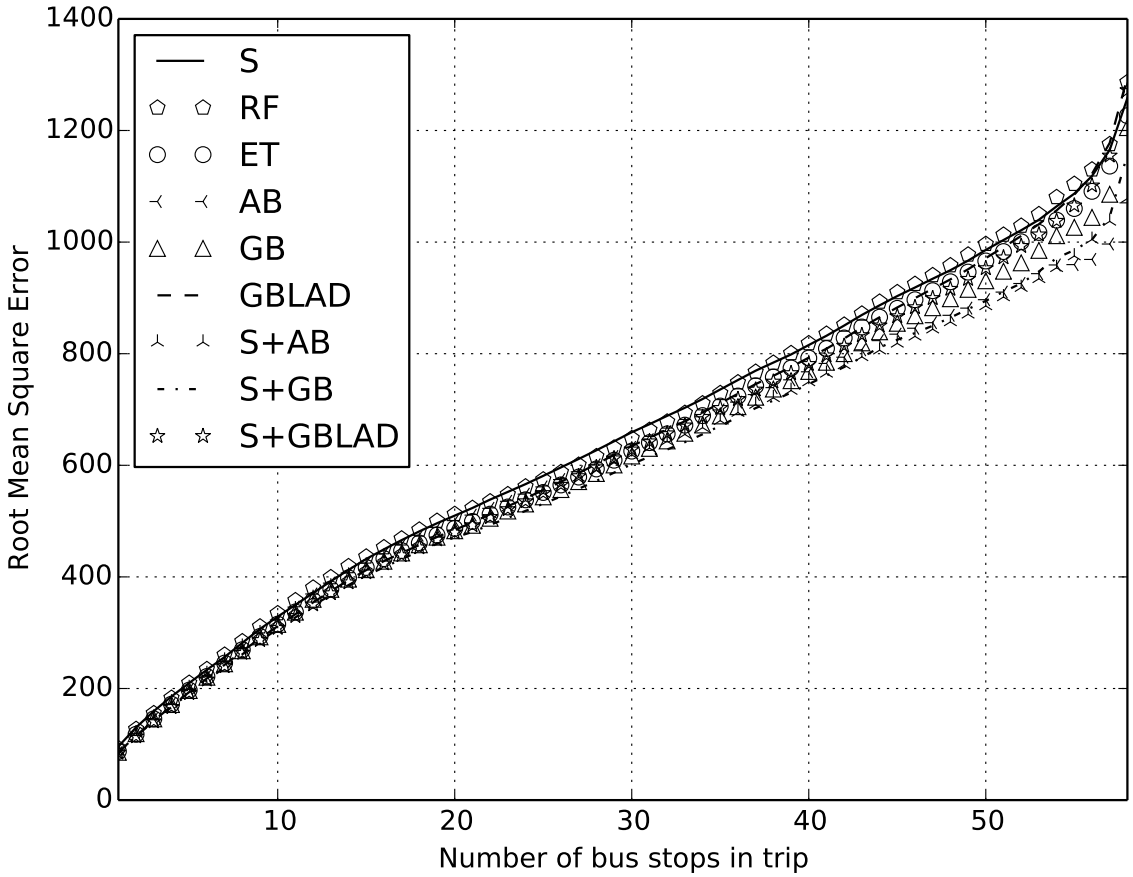


Figure 5: The root mean square error on the duration of the whole trip increases with the trip length. S+GB is the best method for all trip lengths except for length [2,3], [44,54] and [55,58] where respectively S+GBLAD, S+AB and AB are more accurate.

segment does not influence the median of the square estimation error for the snapshot-based methods. This implies yet again that the performance of the snapshot predictor (and its combination with Machine Learning techniques) does not deteriorate (in proportion to the traveling time).

6.3.2 Part 2: Single-Segment Analysis

We start by comparing the results for single-segment predictions of the first stage across methods (Table 4). Unsurprisingly, we observe that additional data in the training set slightly improves the predictive power of all techniques, except for the snapshot prediction, which is a non-learning method. Further, the Table 4 shows that going from a single bus line

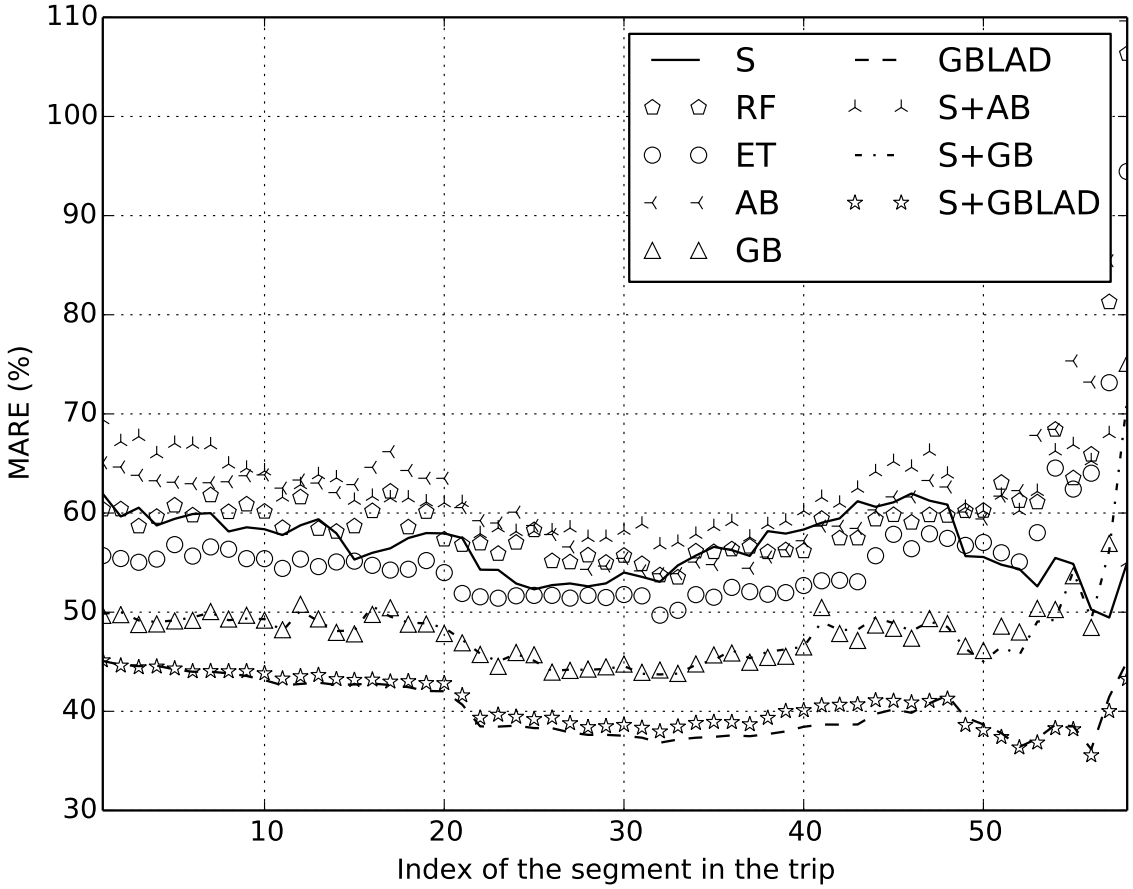


Figure 6: The mean absolute relative error on the duration of a single segment

to four bus lines does not deteriorate the performance of our methods, thus demonstrating the generality and stability of our results. The snapshot predictor performs worse for the second part. Moreover, we observe that all gradient boosting techniques (GB, S+GB, GBLAD, S+GBLAD) coincide, as well as the two non-boosting learning techniques (ET, RF). We verified that for time-varying accuracy measures, these methods yield similar values. However, the two AdaBoost techniques (AB, S+AB) deteriorate between the first and second parts. Therefore, when analyze additional effects (time-of-day and load), we shall demonstrate the effects on the snapshot predictor (S), gradient boosting method (GB), and random forests (RF).

We start with the *time-of-day* effect, which is expected to be influential in transportation systems. Figure 9 corresponds to prediction error (measured in RMSE), as function of

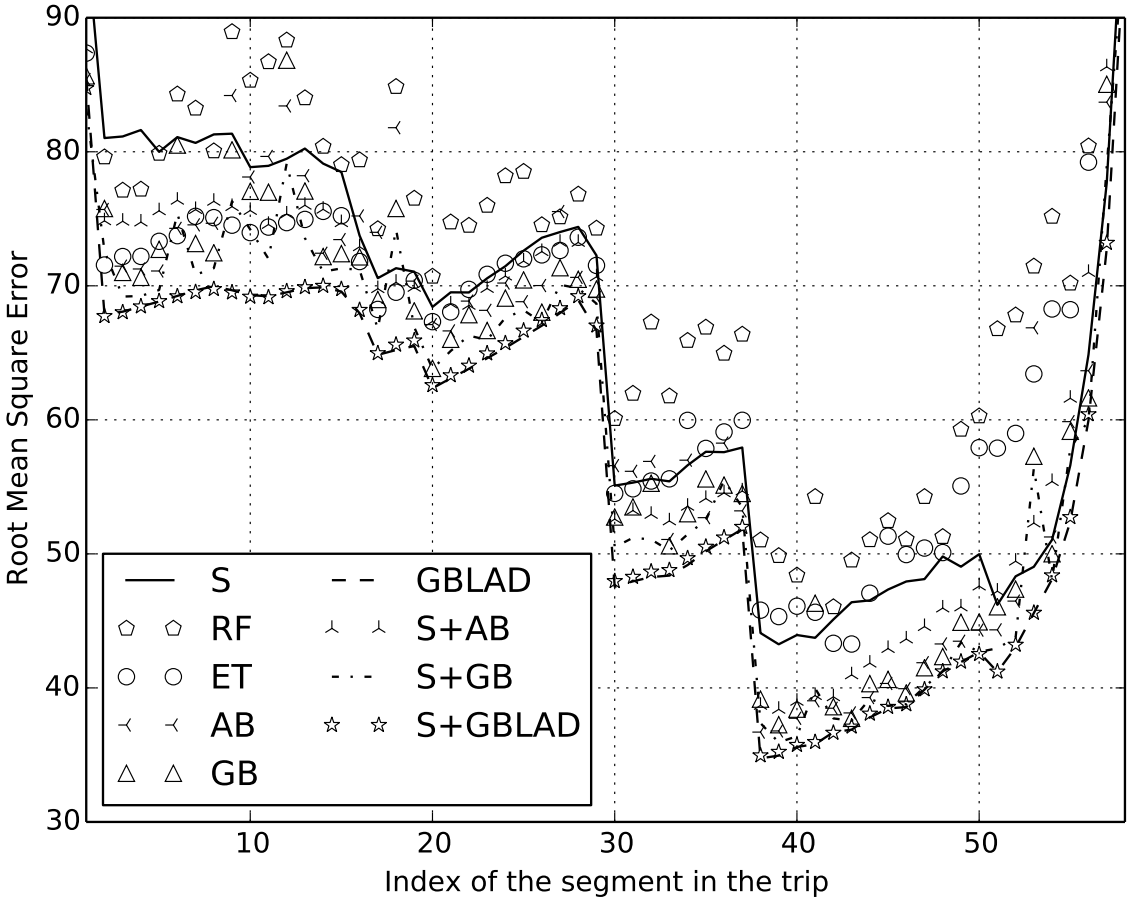


Figure 7: The root mean square error on the duration of a single segment

Table 4: Accuracy of the prediction of the trip length, for the different methods tested over all trips

	S	RF	ET	AB	GB	GBLAD	S+AB	S+GB	S+GBLAD
RMSE (Part 1)	96	83	83	**84**	85	85	**88**	83	84
MARE (Part 1)	61.94	49.45	50.83	60.36	48.88	43.57	69.16	49.30	43.95
MdARE (Part 1)	40.00	30.02	30.85	32.70	29.39	30.23	40.00	29.48	29.45
RMSE (Part 2)	101	**74**	**73**	96	**75**	**74**	102	**75**	**77**
MARE (Part 2)	77.30	62.42	64.56	123.44	62.77	50.85	97.01	63.40	53.00
MdARE (Part 2)	40.00	28.51	29.89	42.86	28.44	28.23	35.59	28.75	28.14

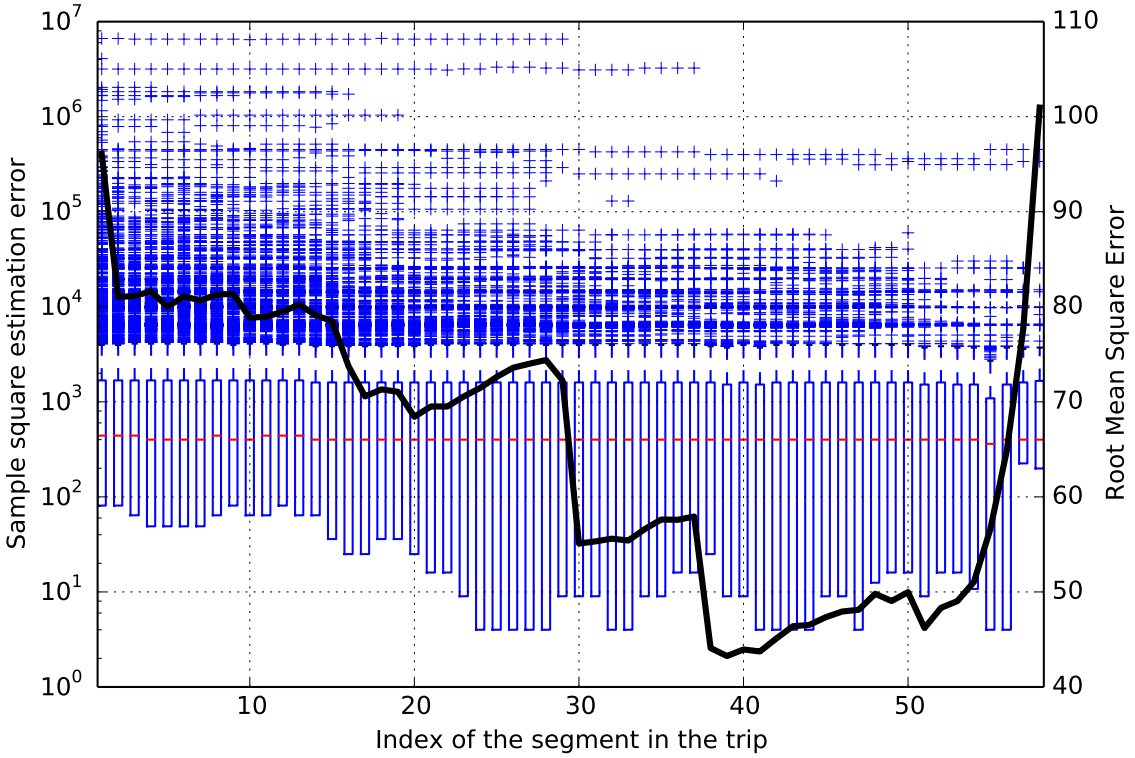


Figure 8: Comparing the boxplots of sample errors (left) to the MSE (right) of the snapshot method as functions of the index of the segment shows that the big drops in the MSE curve are due to the largest outliers disappearing. In particular, look at indexes 30, 38 and 15 to 18.

time-of-day, over the entire test set. The horizontal axis corresponds to half-hour intervals. We observe that the three learning methods that we present (GB and RF) coincide, and have an increasingly improving and stable performance in the afternoon. The snapshot predictor performs worse for almost all half-hour intervals, except for 10:00AM.

Next, we present the system load (number of traveling buses), as function of time of day in half-hours (Figure 10). We observe that the peak is in the morning, with a monotone decrease toward the evening that starts at 5:30PM. Note that the number of traveling buses is only a proxy to system load, since we do not observe the number of cars on the streets, nor the number of passengers waiting for the bus.

Lastly, we correlate the performance of our predictors (RMSE) to the *load*, which we define as the number of buses traveling at a certain time of the day. We present three linear regression lines fitted to the RMSE as function of the load (per half-hour). The results

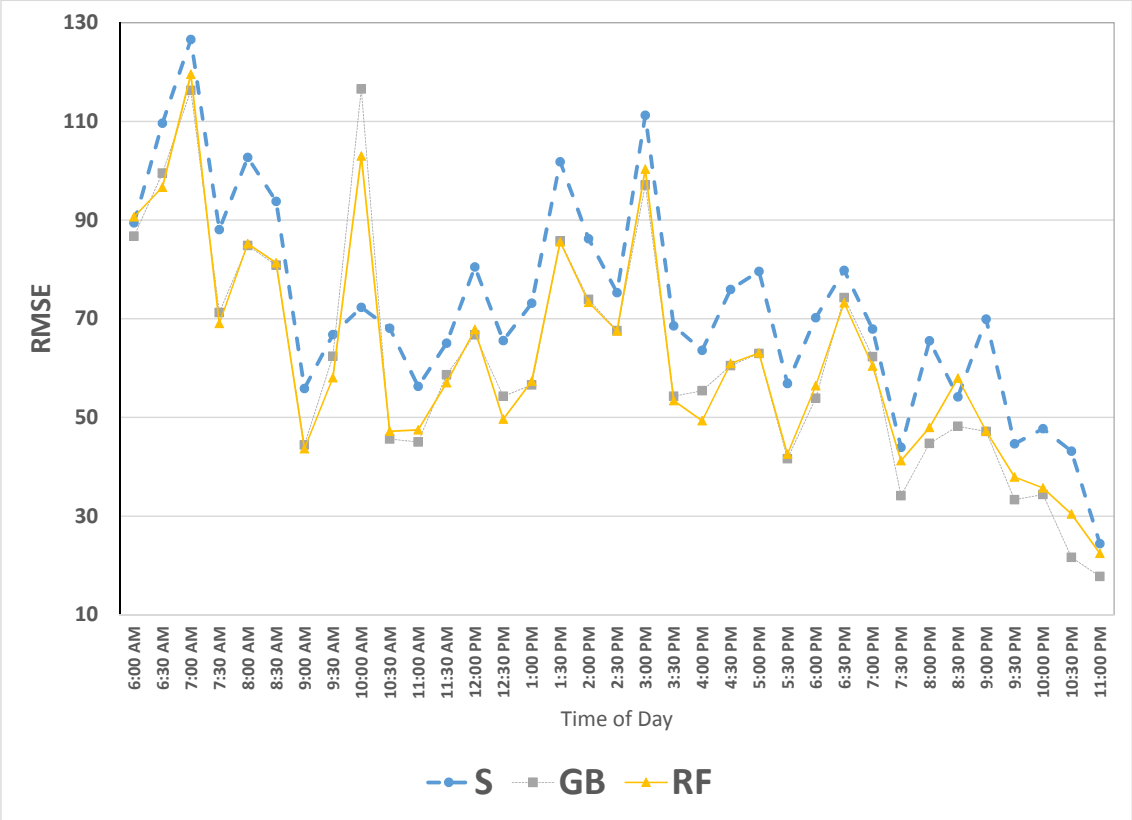


Figure 9: RMSE as function of the time-of-day.

show that the snapshot predictor is most correlated with our proxy of the load ($R^2 = 0.35$), while Random Forests that are not combined with the snapshot principle, present the lowest correlation ($R^2 = 0.27$). Overall, we conclude that the load has a negative effect on the prediction power of our methods. We believe that this is due to the fact that the assumed additive (and uncorrelated) structure of the segmented model is less accurate for high load. For example, the snapshot principle would work better in a stable system, where previous bus rides resemble the current one (evening trips).

7 Related Work

Over the past decade, the problem of predicting traveling times of vehicles, and in particular of buses in urban areas, has received a significant attention in the literature. Most of the work on the subject includes applying various Machine Learning techniques such as

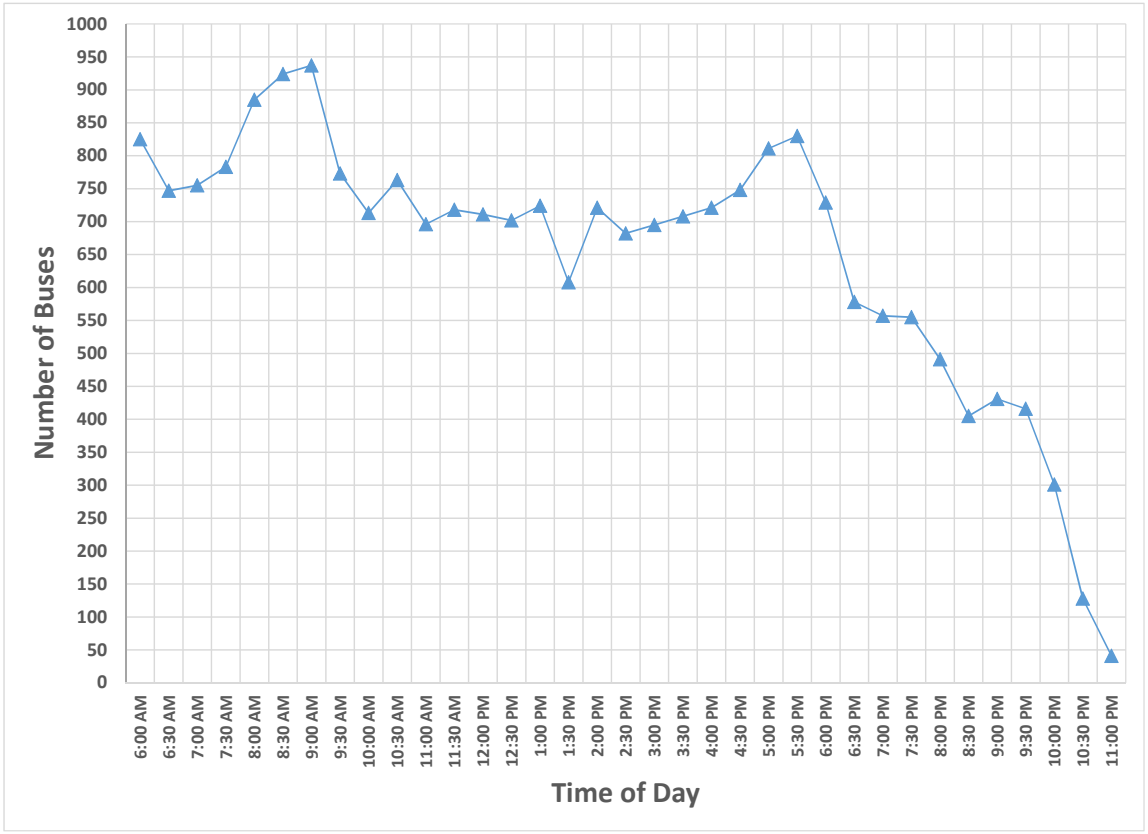


Figure 10: Number of buses traveling (load) as function of the time-of-day.

Artificial Neural Networks [2], Support Vector Machines [1, 4], Kalman Filter models [5], and Non-Parametric Regression models [6]. A thorough literature review of the above techniques can be found in [3].

In recent work, some of the Machine Learning methods were applied to the bus data that we used for the current work, c.f. [7, 8]. Specifically, in [8], Kernel Regression was used on the Dublin bus data in order to predict the traveling time for bus line number $046A$, the same line that we have used for our evaluation. Due to the non-continuous nature of this data (see Section 2), a spatial segmentation of the route into 100-meter segments was proposed. Localizing the bus (with an associated timestamp) is based on GPS measurements that commonly contain large outliers [7], leading for example to many incomplete trips. Such problematic trips are removed from the experiments conducted. In contrast to [8], the segmentation proposed in the current work corresponds to line segments between physical bus stops. Associating timestamps to these stops exploits a data field that relates each

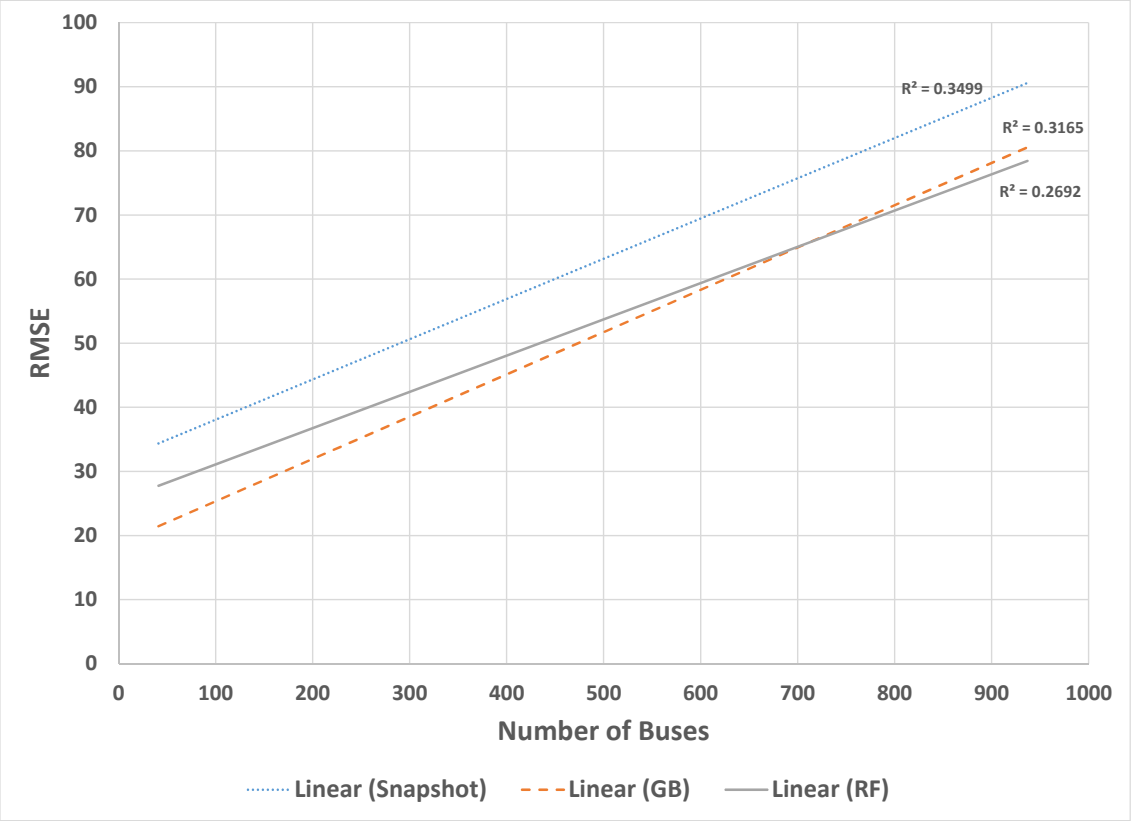


Figure 11: RMSE as function of the number of traveling buses. The corresponding R^2 is presented in proximity to each prediction method.

record of the Dublin bus data to a certain stop. Moreover, to better accommodate for the non-continuous structure of the data, and in alignment with our proposed segmentation, we applied advance learning techniques, *e.g.*, regression trees and boosting.

Traditionally, most state-of-the-art approaches to bus arrival-time prediction consider a single bus line at a time. In [3], Machine Learning models were applied to predict the traveling time of buses to given stops, by using data from *multiple lines* that travel through that same stop. Results show that further information regarding similar bus routes adds value to prediction. In our work, such a multi-line approach is enabled via our model of segmented journeys. Specifically, we take into consideration all bus lines that share bus stops with the journey whose time we aim to predict. To empirically demonstrate the value of the multi-line approach, our evaluation combines traveling times of several bus lines that share stops with line $046A$.

Another contribution of the current paper is the non-learning prediction method based on Queueing Theory. The general application of Queueing Theory to solve transportation problems has been outlined in [9, 10]. However, in most of the works the traffic-flow (*e.g.*, flow of cars) is considered, with traffic modeled as 'customers' in a queueing system. In our work most customers are 'unobserved', while data recordings contain only information on bus travels (i.e. we do not have information regarding cars and other types of transportation). Nonetheless, we apply the *snapshot predictor*, which is a non-learning prediction method that relies on the travel time of the last bus to go through a segment. The motivation for the applications is derived from *queue mining* [11, 12], techniques that come from the domain of business process management, applying Queueing Theory to event data that stems from business processes. The value of these techniques was demonstrated in single-and-multi class queueing systems. Therefore, when considering buses as a class of transportation (that share routes with other classes, *e.g.*, cars), queue mining techniques and specifically, the snapshot predictor, are applicable.

A line of work that combines Queueing Theory and Machine Learning [13, 14] approximates the structure of Web services via queueing networks. Then, the parameters of these networks are estimated from transactional data in low-traffic via Machine Learning techniques such as Monte-Carlo Markov-Chains. Lastly, the results are extrapolated into heavy-traffic scenarios, and are used to asses performance via *e.g.*, response-time analysis. Following this spirit, we propose predictors that integrate the snapshot approach into the regression tree model to create a more accurate prediction method.

8 Conclusion

In this work, we presented a novel approach towards predicting travel time in urban public transportation. Our approach is based on segmenting the travel time into stop-based segments, and combining the use of Machine Learning and Queueing Theory predictors to model traveling time in each segment. Our empirical analysis confirms that the combination of methods indeed improves performance. Moreover, we observe that the snapshot predictor is, counter-intuitively, unaffected by the length of a journey. This leads to positive evidence in favor of applying mixed Queue and Machine Learning predictors in similar settings.

In future work, we intend to extend our methods to support prediction for multi-modal transportation. Also, the mutual support of methods from Queueing Theory and Machine Learning require further investigation to unlock its full potential.

Furthermore, to check the appropriateness of the snapshot predictors as function of car traffic, we intend to add data that comes from a real-time coordinated adaptive traffic system

(SCATS) of Dublin [28]. The system, along with many other functionalities, counts traffic intensity through junctions (in real-time), and records these counts in an event log. We believe that the combination of the two data sets will enable an improvement of prediction, and lead to a better understanding of the root-cause for the errors in our methods.

Moreover, we aim at validating the influence of the *hour-of-day* and *day-of-week* effects that are likely to influence any transportation system. Lastly, we intend to use other data sources (similarly to our use of geographical distances between stops), such as bus-schedules, in order to add valuable features to the Machine Learning techniques, thus reducing prediction error.

Acknowledgment

This work was supported by the EU INSIGHT project (FP7-ICT 318225).

References

References

[1] C.-H. Wu, J.-M. Ho, D.-T. Lee, Travel-time prediction with support vector regression, IEEE Transactions on Intelligent Transportation Systems, 5 (4) (2004) 276–281.

[2] S. I.-J. Chien, Y. Ding, C. Wei, Dynamic bus arrival time prediction with artificial neural networks, Journal of Transportation Engineering 128 (5) (2002) 429–438.

[3] B. Yua, W.H.K. Lama, M.L. Tama, Bus arrival time prediction at bus stop with multiple routes, Transportation Research Part C: Emerging Technologies 19 (6) (2011) 1157 – 1170.

[4] Y. Bin, Y. Zhongzhen, Y. Baozhen, Bus arrival time prediction using support vector machines, Journal of Intelligent Transportation Systems 10 (4) (2006) 151–158.

[5] A. Shalaby, A. Farhan, Prediction model of bus arrival and departure times using avl and apc data, Journal of Public Transportation 7 (1) (2004) 41–62.

[6] H. Chang, D. Park, S. Lee, H. Lee, S. Baek, Dynamic multi-interval bus travel time prediction using bus transit data, Transportmetrica 6 (1) (2010) 19–38.

[7] A. T. Baptista, E. Bouillet, F. Calabrese, O. Verscheure, Towards building an uncertainty-aware personal journey planner, in: ITSC, IEEE, 2011, pp. 378–383.

[8] M. Sinn, J. W. Yoon, F. Calabrese, E. Bouillet, Predicting arrival times of buses using real-time gps measurements, in: ITSC, IEEE, 2012, pp. 1227–1232.

[9] T. Van Woensel, N. Vandaele, Modeling traffic flows with queueing models: a review, Asia-Pacific Journal of Operational Research 24 (04) (2007) 435–461.

[10] G. F. Newell, Applications of queueing theory, Tech. rep. (1982).

[11] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining - predicting delays in service processes, in: CAiSE, Vol. 8484 of LNCS, Springer, 2014, pp. 42–57.

[12] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining for delay prediction in multi-class service processes, Tech. rep., Technion – Faculty of Industrial Engineering and Management (2014).

[13] C. Sutton, M. I. Jordan, Bayesian inference for queueing networks and modeling of internet services, The Annals of Applied Statistics 5 (1) (2011) 254–282.

[14] C. A. Sutton, M. I. Jordan, Inference and learning in networks of queues, in: AISTATS, Vol. 9 of JMLR Proceedings, JMLR.org, 2010, pp. 796–803.

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

[16] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Mining resource scheduling protocols, in: BPM, Vol. 8659 of LNCS, Springer, 2014, pp. 200–216.

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

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

[19] M. I. Reiman, B. Simon, A network of priority queues in heavy traffic: One bottleneck station, Queueing Systems 6 (1) (1990) 33–57.

[20] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and regression trees, Wadsworth International (1984).

[21] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[22] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[23] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning 63 (1) (2006) 3–42.

[24] H. Drucker, Improving regressors using boosting techniques, in: ICML, Vol. 97, 1997, pp. 107–115.

[25] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of Statistics (2001) 1189–1232.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

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

[28] B. McCann, A review of scats operation and deployment in Dublin.

A Creating a Segmented Journey Log

We capture the construction of a segmented journey log as follows. Let (J, α_J) be a J-Log with $\alpha_J = \{\tau, \xi, \pi\}$. Let $j \in J$ be a journey serving journey pattern $\pi(e) = \langle \omega_1, \ldots, \omega_n \rangle$, for all $e \in j$. For a stop ω_i, $1 \leqslant i \leqslant n$, let

$$e_i = \text{argmin}_{e \in j, \xi(e) = \omega_i} \tau(e)$$

be the earliest event for stop ω_i of this journey. Then, for each pair of successive stops ω_i, ω_{i+1}, $1 \leqslant i < n$, a segment event s of schema $\alpha_G = \{\xi_{start}, \xi_{end}, \epsilon, \pi_G, \tau_{start}, \tau_{end}\}$ is created, such that

- $\xi_{start}(s) = \omega_i$ and $\xi_{end}(s) = \omega_{i+1}$,
- $\epsilon(s) = j$,
- $\tau_{start}(s) = \tau(e_i)$ and $\tau_{end}(s) = \tau(e_{i+1})$.

A Segmented J-Log (G, α_G) for (J, α_J) is constructed as a sequence $G = \langle s_1, \ldots, s_m \rangle \in \mathcal{G}^*$ over all segment events of all journeys $j \in J$, such that $\tau_{start}(s_k) \leqslant \tau_{start}(s_{k'})$ for $1 \leqslant k < k' \leqslant n$.

A Segmented J-Log can be trivially constructed from a J-Log. However, in many real-world applications, the recorded data is incomplete due to data loss, unavailability of data

recording devices, or data sampling. Then, it may be impossible to construct a segment event for each pair of successive bus stops of all journeys. For instance, for the data of the bus network in the city of Dublin described in Example 1, due to data sampling, the raw data does not necessarily contain a journey event for each bus stop of the respective journey pattern.

In the remainder of this section, therefore, we outline how to use complementary information on the geographical distances between bus stops and principles of *kinematics* (relating distances to velocity and time) to impute missing journey events and their corresponding timestamps. We assume such distances to be given as a function $\delta : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+$ assigning distance values to pairs of bus stops.

For a journey pattern and a journey recorded in the J-Log, we consider the following three cases of missing sequences:

I) Events missing between two consecutive recorded journey events.
II) Events missing, including the first bus stop.
III) Events missing, including the last bus stop.

We shall first demonstrate the approach to construct the missing journey events for case I) and then demonstrate its refinement for the other two cases.

Let $j = \langle e_1, \ldots, e_u, m_1, \ldots, m_k, e_v, \ldots, e_n \rangle$ be a journey with $m_1, \ldots, m_k$ representing missing journey events, events that according to the journey pattern $\pi(e_1)$ must exist between the stops $\xi(e_u)$ and $\xi(e_v)$. To reconstruct the journey events $m_1, \ldots, m_k$, we need to estimate their timestamps, since information on their route pattern and the respective bus stops are known.

We estimate the timestamps for missing events based on the average velocity $v : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+$ of the respective bus between two stops. It is derived from the events that signal that a bus has been at a certain stop and the distance between the stops:

$$\begin{aligned} v(\omega, \omega') &= \frac{\delta(\omega, \omega')}{\tau(e') - \tau(e)} \\ e &= \text{argmin}_{\hat{e} \in j, \xi(\hat{e}) = \omega} \, \tau(\hat{e}) \\ e' &= \text{argmin}_{\hat{e} \in j, \xi(\hat{e}) = \omega'} \, \tau(\hat{e}) \end{aligned}$$

For journey $j = \langle e_1, \ldots, e_u, m_1, \ldots, m_k, e_v, \ldots, e_n \rangle$ with missing journey events $m_1, \ldots, m_k$, we then assume that the bus travels at a constant velocity through every segment on the way between $\xi(e_u)$ and $\xi(e_v)$, which gives rise to the following recursive approximation of

the timestamps of the journey events:

$$\begin{aligned}\tau(m_1) &= \tau(e_u) + \frac{\delta(\xi(e_u), \xi(m_1))}{v(\xi(e_u), \xi(e_v))}, \\ \tau(m_i) &= \tau_{m_{i-1}} + \frac{\delta(\xi(m_{i-1}), \xi(m_1))}{v(\xi(e_u), \xi(e_v))}, \quad 1 < i \leqslant k.\end{aligned}$$

Next, we target cases II and III, in which the sequence of missing journey events includes the first bus stop, or the last bus stop, respectively. In both cases, we adapt the approach presented above and calculate the velocity of the bus at the segment that still appears in the J-Log after (case II) or before (case III) the recorded journey events, and, as before, assume constant speed of travel throughout the unobserved traveling time. We detail this approach for case II. Case III is handled analogously.

Let $j = \langle m_1, \ldots, m_k, e_u, e_v, \ldots, e_n \rangle$ be a journey with missing events $m_1, \ldots, m_k$. Then, the timestamps of the missing events are determined by the following recursive approximation:

$$\begin{aligned}\tau(m_k) &= \tau(e_u) - \frac{\delta(\xi(m_1), \xi(e_u))}{v(\xi(e_u), \xi(e_v))}, \\ \tau(m_i) &= \tau_{m_{i+1}} - \frac{\delta(\xi(m_i), \xi(m_{i+1}))}{v(\xi(e_u), \xi(e_v))}, \quad 1 \leqslant i < k.\end{aligned}$$

Conformance Checking and Performance Improvement in Scheduled Processes: A Queueing-Network Perspective

Arik Senderovich, Matthias Weidlich, Liron Yedidsion,
Avigdor Gal, Avishai Mandelbaum, Sarah Kadish, Craig A. Bunnell

Abstract

Service processes, for example in transportation, telecommunications or the health sector, are the backbone of today's economies. Conceptual models of service processes enable operational analysis that supports, e.g., resource provisioning or delay prediction. In the presence of event logs containing recorded traces of process execution, such operational models can be mined automatically.

In this work, we target the analysis of resource-driven, scheduled processes based on event logs. We focus on processes for which there exists a pre-defined assignment of activity instances to resources that execute activities. Specifically, we approach the questions of conformance checking (*how to assess the conformance of the schedule and the actual process execution*) and performance improvement (*how to improve the operational process performance*). The first question is addressed based on a queueing network for both the schedule and the actual process execution. Based on these models, we detect operational deviations and then apply statistical inference and similarity measures to validate the scheduling assumptions, thereby identifying root-causes for these deviations. These results are the starting point for our technique to improve the operational performance. It suggests adaptations of the scheduling policy of the service process to decrease the tardiness (non-punctuality) and lower the flow time. We demonstrate the value of our approach based on a real-world dataset comprising clinical pathways of an outpatient clinic that have been recorded by a real-time location system (RTLS). Our results indicate that the presented technique enables localization of operational bottlenecks along with their root-causes, while our improvement technique yields a decrease in median tardiness and flow time by more than 20%.

1 Introduction

Service systems play a central role in today's economies, e.g., in transportation, finance, and the health sector. Service provisioning is often realized by a *service process* [1, 2]. It can be broadly captured by a set of activities that are executed by a service provider and designated to both attain a set of organizational goals and add value to customers.

Independently of the domain, service processes can be classified by the amount of interactions between service providers and customers and the level of demand predictability and capacity flexibility. A service can be *multi-stage*, involving a series of interactions of a customer with a provider, or specific resources at a provider's end. Further, a process can be *scheduled*, meaning that the number of customers to arrive is known in advance, up to last moment cancellations and no-shows. Then, customers follow a schedule, which is a pre-defined series of activity instances, each having assigned a planned starting time for its execution, a duration, and the involved resource.

Multi-stage scheduled processes are encountered, for instance, in outpatient clinics, where various types of treatments are provided as a service to patients [3]. Here, a schedule determines when a patient undergoes a specific examination or treatment. Another example of multi-stage scheduled processes is public transportation, where schedules determine which vehicle serves a certain route at a specific time [4].

In this work, we focus on operational analysis for multi-stage scheduled service processes. Specifically, we aim at answering the following two key questions: *how to assess the conformance of a pre-defined schedule of a service process to its actual execution?* and *how to improve operational performance of the scheduled process?*

To address the first question, we present a method that is grounded in a queueing network for both the schedule and the actual process execution and applies statistical inference (hypotheses testing) and similarity assessment to validate the scheduling assumptions of the process. As outlined in Figure 1, the conformance checking step yields diagnostics on operational deviations between the schedule and the execution of the process. The identified deviations then guide the efforts to improve the operational performance of a process. In particular, we target improvements in terms of decreased tardiness (lateness with respect to due dates) and lower flow time by adapting the scheduling policy.

We base our technique on a generalization of a specific type of *queueing networks*. This choice is motivated by the need to capture two aspects of service processes in particular. First, the key actors of service processes namely, customers and service providers (or resources), and their complex interaction in terms of customer-resource matching policies (e.g. First-Come First-Served, Most-Idling Resource-First) [5] need to be specified. Second, a network model is

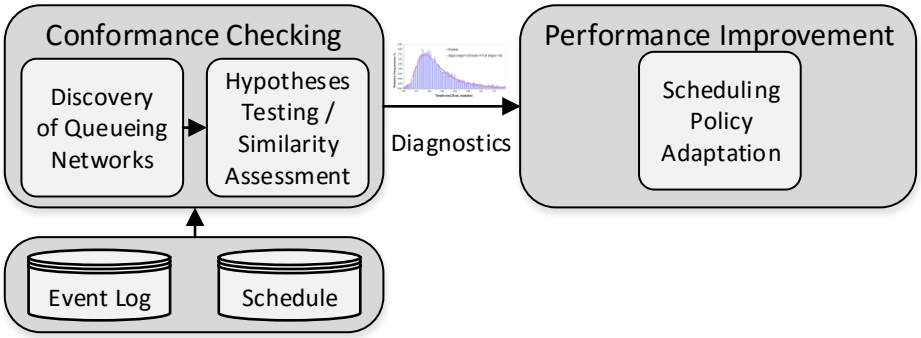


Figure 1: An outline of our approach

required to define the dependencies of different stages of the service process, including parallel processing of activities [6]. Against this background, we rely on *Fork/Join networks* [7], which serve as the foundation for conformance checking and enable performance analysis of parallel queueing systems [8].

Our contributions can be summarized as follows:

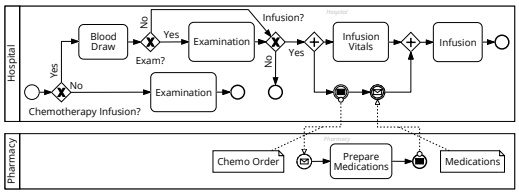
(1) We present a method to assess the conformance of a schedule and the actual process execution based on queueing networks. By means of statistical inference and similarity assessment, we identify operational deviations along with their root-causes in terms of violated assumptions underlying the scheduling mechanism.

(2) We present a process improvement technique that relies on the identified root-causes to adapt the scheduling policy of the service process to decrease the tardiness and lower the flow time.

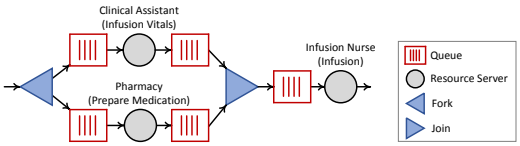
This paper is an extended and revised version of our earlier work that focused on conformance checking in scheduled processes [9]. In this work, we improve, extend and formalize the earlier proposed model validation technique. Furthermore, we complement the conformance checking approach with a process improvement technique.

We demonstrate the value of the proposed approach by a two-step evaluation. First, we apply the conformance checking techniques to RTLS-based data from a real-world use-case of a large outpatient oncology clinic namely, the Dana-Farber Cancer Institute.[1] Our experiments demonstrate the usefulness of the extended validation method for detection of operational deviations and identifying root causes for them. As a second evaluation step, we present simulation-based experiments that evaluate the proposed process improvement technique and show that tardiness and flow time can be reduced by more than 20% using the adapted scheduling policy.

[1] `http://www.dana-farber.org/`.



(a) The control-flow perspective of patient flow in the day hospital.



(b) A Fork/Join network that depicts the scheduled process from the resource perspective..

Figure 2: Patient flow in Dana-Farber Cancer Institute.

The remainder of the paper is structured as follows. The next section presents a detailed use-case of a process in an outpatient clinic to motivate our approach. The models for the service process data, specifically, the schedule and the event log, are presented in Section 3. Fork/Join networks and their discovery from data are discussed in Section 4, before we turn to the method to assess conformance in Section 5. Section 6 introduces an approach for improving the operational performance of a service process, which is guided by the diagnostics obtained by conformance checking. An empirical evaluation of our approach, based on real-life data logs and trace-based simulation is given in Section 7. Section 8 discusses related work, followed by concluding remarks (Section 9).

2 A Service Process in an Outpatient Clinic

We illustrate the challenges that arise from operational analysis of multi-stage scheduled service processes through a process in the Dana-Farber Cancer Institute (DFCI), a large outpatient cancer center in the US. In this hospital, approximately 900 patients per day are served by 300 health care *providers*, e.g. physicians, nurse practitioners, and registered nurses, supported by approximately 70 administrative staff. The hospital is equipped with a Real-Time Location System (RTLS). We use the movements of patients, personnel, and equipment recorded by this system to evaluate our approach.

We focus on the service process for a particular class of patients, the on-treatment patients (OTP). This process applies to 35% of the patients, yet it generates a large fraction of the workload due to the long processing times. Hence, operational analysis to balance quality-of-service and efficiency is particularly important for this process. Figure 2a depicts the control-flow perspective of the process as a BPMN diagram: arriving patients may directly receive examination by a physician, or shall undergo a chemotherapy infusion. For these patients, a blood draw is the initial appointment. Then, they either move to the infusion stage directly, or first see a provider for examination. Infusions may be cancelled,

after examination or after measuring infusion vital signs.

For a specific scheduled part of the aforementioned chemotherapy infusion process, Figure 2b illustrates a queueing network that captures the resource perspective of the process. This model is a Fork/Join network, discussed in more detail in Section 3. It represents the associated resources: clinical assistants, a pharmacy, and infusion nurses, as well as dependencies between them that follow from the patient flow. Patients first fork and enter two *resource queues* in parallel: one is the queue where they actually sit and wait for a clinical assistant to take their vital signs; the other queue is virtual, where they wait for their chemotherapeutic drugs to be prepared by the central hospital pharmacy. The process can only continue once both of these parallel activities are completed, which explains the existence of a *synchronization queue* in front of the join of the flows. After the join, patients are enqueued to wait for a nurse and chair to receive infusion.

The provisioning of infusions (as well as other procedures) in DFCI is scheduled. Specifically, each patient has a schedule that assigns a planned start time, duration and resource type to the respective activity instance, also referred to as a task. In the presence of a recorded event log, one may consider two centric types of performance-related questions. The first question is about conformance of the schedule and process execution, i.e., does the *planned* execution of the process corresponds to the recorded reality. This question has vast implications on operational considerations. Specifically, staffing of service providers, and information released to patients and physicians are governed by the schedule. Therefore, it is important that the schedule functions as an appropriate proxy to the real process. A second question inquires as to how to improve the operational process performance based on the insights obtained from an analysis of the event log. Motivated by the DFCI use-case, our work provides a novel approach to analyse these two questions.

3 Schedules and Event Logs of Service Processes

In this work, we provide a multi-level analysis approach that exploits two types of input data, namely a *schedule* and an *event log* of recorded tasks, i.e., actual executions of activities. Below, we first introduce a running example that is based on the DFCI use-case, to give some intuition for our definitions. Then, we formalize the models of a schedule and an event log.

3.1 Running Example

Consider two patients that are scheduled to visit the Dana-Farber Cancer Institute, and receive chemotherapeutic treatment. The first patient, id number 111, is scheduled to go through a blood draw procedure, a physician's examination, and a chemotherapy infusion. The second patient, id number 222, is planned to perform only a blood draw and a chemotherapy infusion, and does not require an examination prior to the infusion.

In reality, patient 111 went through vital signs activity, prior to receiving infusion. Vital signs is not a scheduled activity, i.e., there is no task that would include a pre-defined start time, resource type, or planned duration. Also, the preparation of a chemotherapeutic drug is performed in the DFCI pharmacy during the visit. This stage is also unscheduled. However, since the infusion itself is scheduled, these two activities (vitals and drug production) must end prior to the scheduled execution of the infusion. The second patient went through the blood draw stage and had the infusion cancelled, due to inadequate blood results.

The full detail of the example is captured in the two data logs, the schedule and the event log presented in tables 1 and 2.

3.2 Schedule

A schedule (e.g. Table 1) represents the plan of a multi-staged service process for individual customers, which is comprised of partially ordered *tasks*. We define a task to be a relation between case identifiers, activities, and resources at a given time for a certain duration. In our running example, customer with a case identifier 111 is to perform an *infusion* procedure with an *infusion nurse*, which is scheduled to 10:30, and is planned to last 180 minutes. We denote the universe of tasks by T (including the empty task ϵ), and the set of resource types, or roles (e.g. infusion nurse) by R. We assume that activities (from universe of activities A) can be performed by a single resource type at a time (no shared resources). However, activities can be performed by several resource types (blood draw can be performed by nurse or a phlebotomist), and resource types can perform several activities.

Definition 1 (Schedule) *A* schedule *is a set of planned tasks, $T_P \subseteq T$, having a schema (set of functions) $\sigma_P = \{\xi_p, \alpha_p, \rho_p, \tau_p, \delta_p\}$, where*

- $\xi_p : T \to \Xi$ *assigns a case identifier to a task.*
- $\alpha_p : T \to A$ *assigns an activity to a task.*
- $\rho_p : T \to R$ *assigns a resource type to a task.*
- $\tau_p : T \to \mathbb{N}^+$ *assigns a timestamp representing the* earliest *start time to a task (e.g. in UNIX time).*

Case Id	Activity	Resource Type	Start Time	Duration
111	Blood Draw	Phlebotomist	7:30:00	15MIN
111	Exam	Physician	9:30:00	30MIN
111	Chemo. Infusion	Inf. Nurse	10:30:00	180MIN
222	Blood Draw	Nurse	9:30:00	15MIN
222	Chemo. Infusion	Inf. Nurse	11:00:00	120MIN

Table 1: Example from schedule of Dana-Farber Cancer Institute

Case Id	Activity	Resource Type	Start Time	Duration
111	Blood Draw	Phlebotomist	7:12:00	8MIN
111	Exam	Physician	9:30:00	42MIN
111	Vitals	C. Assistant	10:12:00	4MIN
111	Chemo. Product.	Pharmacy	10:12:00	35MIN
111	Chemo. Infusion	Inf. Nurse	10:47:00	167MIN
222	Blood Draw	Nurse	9:26:00	25MIN

Table 2: Example from an event log of Dana-Farber Cancer Institute

- $\delta_p : T \to \mathcal{D} \subseteq \mathbb{N}^+$ *assigns a duration to a task (e.g. pre-defined set of times in minute units).*

The timestamp (τ_p) and duration (δ_p) assignments induce a partial order of tasks, denoted by $\prec_P \subseteq T_P \times T_P$.

3.3 Event Log

An event log (e.g. Table 2) contains the data recorded during the execution of the service process, e.g., by a Real-Time Location System (RTLS) as in our use-case scenario (Section 2). Tasks in the log relate to a customer, a resource, an activity, and timestamps of execution, thereby representing a unique instantiation of an activity executed by a resource for a customer at a certain time.

Definition 2 (Event Log) *A* log *is a set of executed tasks, $T_A \subseteq T$, having a schema $\sigma_A = \{\xi_a, \alpha_a, \rho_a, \tau_{start}, \tau_{end}\}$, where*

- $\xi_a : T \to \Xi$ *assigns a case identifier to a task.*
- $\alpha_a : T \to A$ *assigns an executed activity to a task.*
- $\rho_a : T \to R$ *assigns a resource type that executed the task.*
- $\tau_a : T \to \mathbb{N}^+$ *assigns a timestamp representing the* observed *start time to a task.*
- $\delta_a : T \to \mathbb{N}^+$ *assigns the actual duration to a task (e.g. in minutes).*

The timestamps and durations assigned by τ_a, δ_a induce a partial order of executed tasks, denoted by $\prec_A \subseteq T_A \times T_A$.

There are differences between planned and actual schema functions. For example, for patient number 111, vital signs and chemotherapy preparation activities do not appear in the

schedule, yet occur in reality. Also, we allow for scheduled activities to be cancelled for some of the process instances. However, we assume that the recorded event log has the following property.

Property 1 (Resource Inclusion) *Let R_p be the image of ρ_p, and R_a be the image of ρ_a. Then, $R_p \subseteq R_a \subseteq R$.*

This property ensures that resource types appearing in the schedule also appear (at least once) in the event log. The other direction does not necessarily hold, as unscheduled activities can be performed by resource types that do not appear in the schedule.

As a final comment, we assume the existence of an injective function $\nu : T_A \to T_P$, that maps actual tasks to their scheduled counterparts. The empty task ϵ is assumed to be included in T_P to allow for unplanned activities. That is, if for some executed task, $t \in T_A$, there exists a scheduled task t', then $\nu(t) = t'$ holds. If the task t has not been scheduled, then $\nu(t) = \epsilon$.

4 Fork/Join Networks: Definition and Discovery

We base the conformance checking and performance improvement in scheduled processes on a general family of queueing networks, namely Fork/Join networks (F/J networks). F/J networks, in contrast to ordinary queueing networks, capture both resource delays (due to a lack of available resources to execute a certain activity) and synchronisation delays (due to concurrent activities that have not finished execution). In addition, F/J networks naturally support service policies that govern how resources are assigned to instances of a service process (e.g., First-Come First-Served). In contrast, behavioural formalisms such as Petri-nets require extensions that hinder their analysis to express such policies [10]. Finally, there is a rich body of techniques for F/J networks that approximate optimal service policies and analyze time behaviour of service processes [8, 11].

We start with a formal definition of F/J networks that will serve us in the current work (Section 4.1). Then, in Section 4.2 we elaborate on the discovery of F/J networks from event logs. Although a full-fledged automatic discovery of Fork/Join networks from data is beyond the scope of this paper, we outline how existing ideas from process mining are combined with basic statistics to obtain an initial F/J network, which is then completed manually.

4.1 Fork/Join Networks: Definition

Formally, queueing networks are directed graphs, with vertices corresponding to server nodes (or service providers). Instances of a service process (*customers* hereinafter) traverse through

services that are performed by servers, according to probabilistic routing [12].

Fork/Join (F/J) networks supports splitting and joining of customers, which makes them particularly suitable to model concurrent processing [11]. F/J networks support two types of queues: *resource queues* are formed due to limited resource capacity, and *synchronization queues* result from simultaneous processing by several resources. Servers can thus be of three types, namely (1) regular (resources with finite or infinite capacity), (2) fork, and (3) join.

In this work, we consider *open* F/J networks (customers arrive from outside the system and depart eventually) that exhibit *multi-class* services (servers execute several activities), and *probabilistic choices*. This formalism is inspired by the conceptual framework of *processing networks* [13, 14], and generalizes both multi-class networks [12] and Fork/Join networks [11].

In multi-class queueing networks, customers that arrive to a server may encounter different types of processing. Examples for such types of processing include not only the execution of different activities, but also different ways of executing an activity. For instance, in scheduled processes, activities may be planned with different durations, so that each combination of an activity and its planned duration is treated differently in the scheduling of resources.

Formally, this aspect is captured by a set of *customer classes*, which we denote by $C \subseteq \mathbb{N}^+$. The relation between tasks and customer classes is established by a function $\psi : T \to C$. Taking up the running example, a physician may perform two activities, examination and consultation. However, examinations can be planned for 15 minutes or 30 minutes and, depending on the duration, the scheduling is implemented differently. As such, in this example, there are three customer classes (consultation, examination in 15 minutes, examination in 30 minutes).

To define F/J networks, we need to specify *server dynamics* for each of the servers in the net. To this end, we adopt a version of Kendall's notation [15], so that every server is characterized by five building blocks, $\mathcal{A}_t/\mathcal{B}_c/\mathcal{R}_t/\mathcal{Z}/\mathcal{P}$ where $\mathcal{A}_t$ represents the (time-varying) *external arrival process* that are class independent; $\mathcal{B}_c$ corresponds to (stationary) *processing time distribution* for customer class $c \in C$ (time-independence is typically assumed for service processes, see [16]); and $\mathcal{R}_t$ stands for time-changing resource *capacity*, i.e., the number of resources working in the server node at time t. The *class assignment* function, $\mathcal{Z}$, assigns a class to an arriving customer with probability $Z(c)$, where $\sum_{c \in C} Z(c) = 1$.

Component $\mathcal{P}$ is the stationary *service policy* that sets both the order of entry-to-service, among the enqueued customers, and selects the resource, among available ones, to serve a customer. For example, the most well-known service policy for queues is the First-Come First-Served (FCFS) policy. Resources related to server nodes are work-conserving (immediately engaging in service when available) and statistically identical with respect to the distribution

of their processing times. All five building blocks are assumed to be independent of one another.

With $\mathcal{K}$ being the universe of possible dynamics models for a server, we define F/J network as a probabilistic network of three different types of server nodes, each server being assigned a dynamics model.

Definition 3 (Fork/Join Network) *A* Fork/Join network $\mathcal{F}$ *is a triple* $\langle S, W, b \rangle$*, where*

- $S = S_R \cup S_F \cup S_J$ *is a set of servers, with* S_R *being a set of resource types and* S_F, S_J *being sets of forks and joins, respectively such that* $S_R \cap S_F \cap S_J = \emptyset$*;*
- $W : (S \times S) \to [0, 1]$ *is a routing matrix (or the weighted flow) between servers;*
- $b : S \to \mathcal{K}$ *assigns a dynamics model to servers.*

We consider forks and joins to be zero-delay and zero-capacity resource nodes. The weights of arcs coming out (in) of a fork s_f (a join s_j) are assumed binary, that is, a customer always routes to (from) a downstream (upstream) server, s', and thus $W(s_f, s') = 1$ ($W(s', s_j) = 1$).

The weights of the routing between servers correspond to probabilities. Therefore, $0 \leq W(s, s') \leq 1, \forall s, s' \in S$ and $\sum_{s' \in S} W(s, s') = 1$.

As an example, consider the F/J network in Figure 2b. This visualisation contains, in addition to server nodes and routing, three resource queues (preceding resources) and two synchronization queues (succeeding resources). It is worth noting that an F/J network is *fully characterised* by servers, routing, and server dynamics. Queues are defined implicitly before and after their respective servers. The example in Figure 2b contains three resource types, a fork, and a join. For each resource type, there is a single customer class for the activity performed by the resources of the respective type.

4.2 Discovery of Fork/Join Networks

Discovery of a F/J network from data, either a schedule or an event log, involves the identification of the network structure (S), an estimation of the routing (W), and a characterisation of server dynamics (b). Below, we outline how existing process mining techniques can be used in our setting to create an initial F/J network as the basis for manual refinement.

Discovery of Structure and Estimation of Routing. To discover the general structure of a network, a large variety of existing process mining techniques can be used (see [17, Ch. 5] and the references within). In particular, given that the schedule and the event log both contain resource types and start times of tasks as well as their durations, we follow an approach that resembles the time-interval-based process discovery proposed by

Burattin [18]. Specifically, we employ interval algebra [19] and assume that resources of different types that perform tasks concurrently at least once are indeed concurrent in the network. Subsequently, the required forks and joins are inserted, prior to resource nodes with more than one predecessor or successor, respectively. This yields a 100% fitting model, in the sense that all resource types and possible routing paths are represented.

For practical reasons, we also add input and output resource nodes with empty dynamics: an input node $s_i \in S_R$, to which all customers arrive (externally), and which is connected to all servers with external arrivals; an output node $s_o \in S_R$ that is connected to all server nodes in which the process terminates according to the data.

To estimate the flow matrix W, we start by assuming that all edges between the nodes are deterministic, and all weights are equal to 1. For edges that leave a fork or connect to a join, this weight remains unchanged. For all other edges between servers s and s' (s' can be a fork), the weights are set to Markovian probabilities using an estimator that is calculated as the number of occurrences of the transition from s to s' in the data divided by the number of occurrences of s.

Lastly, weights for the input and output resource nodes are set as follows. The weight for a transition from s_i to s is set to the proportion of customers that arrive at node s out of all exogenous arrivals. Similarly, the weight for a transition from s to s_o is set to the proportion of services that terminated in s.

Characterising Server Dynamics. For most of the components of server dynamics, mining techniques have been presented in the literature. An exception is the derivation of the customer classes C, needed to extract the distribution of processing times per class. The customer classes are server-dependent, yet may be defined based on various properties of tasks. In most cases, taking the activities executed by resources of a particular type is a reasonable initial definition of the customer classes. However, manual refinement may be needed to extract, for instance, combination of activities and durations that define a class. Given a particular notion of classes, the probabilities $Z(c)$ are estimated per server $s \in S_R$ by the number of customers falling into class c, out of all visits in s, as recorded in the data.

Once $Z(c)$ are estimated, the remaining components of server dynamics are derived using existing techniques. The number of resources as function of time, $\mathcal{R}_t$, is extracted from data using statistical methods as reported in [20]. The distribution of inter-arrival times $\mathcal{A}_t$ and processing times per class $\mathcal{B}_c$ can be fitted with the techniques presented in [16, 21]. Service policies, $\mathcal{P}$, can be discovered using the policy-mining techniques presented in [22], or assumed to be given, as in the case of discovering a F/J network from a schedule.

5 Conformance Checking in Scheduled Processes

This section introduces an approach to assess the conformance of a pre-defined schedule of a service process to its actual execution. Following the existing theory for validating (simulation-based) operational models against execution data [23], we decompose the conformance checking problem along two dimensions, namely conceptual and operational.

Conceptual conformance checks the assumptions and theories that underlie the schedule. That is, we compare the schedule and the event log indirectly by means of F/J networks that are discovered for both. These networks are compared through the lenses of their corresponding components: structure, routing, and server dynamics, which enables general insights beyond the level of instance-based conformance checking.

Operational conformance checks the 'predictive power' of a schedule with respect to various performance measures (e.g., delay predictions). To this end, based on the schedule and the event log, we measure deviations between the observed and the scheduled performance indicators.

Conceptual and operational conformance are often linked in the sense that issues in operational conformance can be explained by a lack of conceptual conformance. For instance, if operational conformance checking identifies that queues are shorter or longer than planned, this may indicate an overstaffed or an understaffed system, respectively, which is a problem of conceptual conformance.

This section first presents our methodology for conceptual and operations conformance checking (Sections 5.1-5.2). We then tie the two conformance types and elaborate on how to detect performance deviations from the schedule and identify root-cause explanations of deviations (Section 5.3).

5.1 Conceptual Conformance to Schedule

Given two F/J networks $\mathcal{F}_P$ and $\mathcal{F}_A$ that have been discovered from a schedule ($\mathcal{F}_P$, for *planned*) and an event log ($\mathcal{F}_A$, for *actual*), respectively, conceptual conformance checking compares their components. Below, we first present a methodology for this comparison, which is followed by a discussion of the specific algorithms to instantiate the methodology.

Comparing Components of F/J Networks. The components of the log-based F/J network $\mathcal{F}_A$ can be either stochastic or deterministic. For example, processing times are often assumed to be stochastic, while service policies and resource capacities are typically assumed deterministically pre-defined. The schedule-based network $\mathcal{F}_P$ is entirely deterministic. Therefore, we need to consider a comparison framework that includes two types of

comparisons, namely stochastic to deterministic (S2D), and deterministic to deterministic (D2D).

A natural framework for S2D comparison is statistical hypothesis testing [24]. Here, the stochastic element is summarized as a distribution, either parametric or non-parametric, and its parameters (or quantiles for non-parametric distributions) are compared to hypothetical values that correspond to the schedule. For example, the duration of an activity can be characterized by a non-parametric distribution. Hypothesis testing then verifies whether the median of the distribution is indeed equal to the planned duration of this activity, thereby comparing the scheduled and actual processing times.

The null hypothesis for S2D comparisons is that the components of $\mathcal{F}_P$ and $\mathcal{F}_A$ are equal. To test the hypothesis, a test statistic is constructed from the event log, such that its distribution under the null hypothesis is known. It is computed as a function of the measurements in the event log (samples), and given these measurements one can derive the probability under which the difference between the actual and the scheduled is significant, thus causing rejection of the null hypothesis.

Hypothesis testing is not suitable for D2D comparisons, since *any* deviation between the realizations in the event log and the scheduled counterpart will result in rejection of the null hypothesis. Hence, we resort to measures of similarity and/or dissimilarity that are specifically developed for the components of F/J networks.

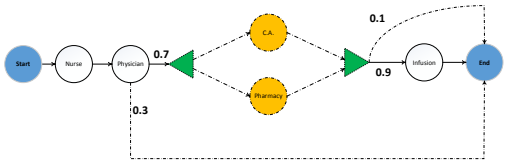
Structural Conformance. Given the F/J networks $\mathcal{F}_P = \langle S_P, W_P, b_P \rangle$, $S_P = S_{R_P} \cup S_{F_P} \cup S_{J_P}$, and $\mathcal{F}_A = \langle S_A, W_A, b_A \rangle$, $S_A = S_{R_A} \cup S_{F_A} \cup S_{J_A}$, structural conformance (a D2D comparison) relates to the resource nodes S_{R_P} and S_{R_A}, respectively. Fork and join nodes (S_{F_P}, S_{J_P}, S_{F_A}, S_{J_A}) are not considered as any difference related to concurrent execution of activities is part of the routing conformance, i.e., the comparison of W_P and W_A.

The sets S_{R_P} and S_{R_A} are not necessarily equal, since there can be unscheduled activities, performed by resource types that are not acknowledged in the schedule. We assess the difference of the sets of resource nodes by means of the intersection set $S_I = S_{R_P} \cap S_{R_A}$, which is always equal to S_{R_P} due to Property 1, and the difference set $S_D = S_{R_A} \setminus S_{R_P}$. Then, a natural measure for the similarity of S_{R_P} and S_{R_A} is defined as the number of resource nodes in the schedule-based model in relation to all resource nodes observed in the event log:

$$\varphi(S_{R_A}, S_{R_P}) = \frac{|S_I|}{|S_{R_A}|}. \tag{1}$$

This measure reaches its maximum of 1, whenever the sets of resource nodes are equal. The minimum value of 0, in turn, indicates that there are no planned activities according to

(a) An example of a scheduled process - a F/J net with binary weights on arcs.



(b) An example for the actual counterpart of the F/J net in Figure 3a.

Figure 3: Scheduled and Actual F/J network structures.

schedule. The related measure for dissimilarity is

$$d(S_{R_A}, S_{R_P}) = \frac{|S_D|}{|S_{R_A}|}, \tag{2}$$

which is exactly $1 - \varphi(S_{R_A}, S_{R_P})$, since $S_{R_A} = S_D \cup S_I$.

While both measures play a minor role as a stand-alone comparison between F/J networks, the underlying sets S_I and S_D assume important roles for testing conformance of the routing and the server dynamics of two networks. Specifically, assumptions related to routing and server dynamics are tested solely for the resource nodes in both networks (S_I). In addition, the dynamics of the resource nodes in S_D become relevant in the context of process improvement (Section 6).

Taking up the running example, Figures 3a and 3b present network structures including routing that are discovered from a schedule and an event log, respectively. The intersecting resource nodes S_I are Nurse, Physician and Infusion Nurse, whereas Clinical Assistant (C.A.) and Pharmacy belong to S_D, and it holds that $\varphi = 0.6$.

Routing Conformance. For routing conformance, we compare the matrices W_P and W_A. This is an S2D comparison, with a stochastic W_A and a deterministic W_P . The null hypothesis to test is whether W_P corresponds to W_A. That is, for every resource node, we compare the actual probability to leave it into the scheduled resource node. We consider $|S_I|$ null hypotheses and alternate hypotheses, defined for $s \in S_I$ as:

$$\begin{aligned} H_{0,s} &: W_A(s, s') = W_P(s, s'), \forall s' \in S_I \\ H_{1,s} &: \textit{otherwise}. \end{aligned} \tag{3}$$

Since we are interested only in resource nodes the weights $W_A(s, s')$ are probabilities and the routing matrix W_A corresponds to a Markov chain. Given a sample of $W_A(s, s')$ from the event log we can then perform a χ^2 test for Markov chains, to accept or reject $H_{0,s}$ for

each $s \in S_I$ [25]. Resource nodes that 'fail' the test—the corresponding null hypothesis will be rejected at a certain significance level—are the root-cause for conceptual deviations in the routing.

Intuitively, we would like to use the tasks T_A of the event log as the sample of W_A. However, it has to be ensured that only tasks executed by resource nodes that are part of the schedule are considered. Hence, solely the tasks in $T_I = \{t \in T_A \mid \rho_a(t) \in S_I\}$ are used as the sample to construct W_A.

Two resource nodes of special interest are s_i (input) and s_o (output). Deviations related to s_i correspond to unscheduled external arrivals, whereas deviations related to s_0 are exceptions in exiting the system.

Dynamics Conformance: External Arrivals. To assess conformance of server dynamics, each of the building blocks $\mathcal{A}_t/\mathcal{B}_c/\mathcal{R}_t/\mathcal{Z}/\mathcal{P}$ have to be checked. Starting with the external arrivals, we assess the *tardiness* of arrivals, i.e., the deviation between planned arrival times and the measured arrivals [26].

We divide the analysis of tardiness into three parts. First, we propose a stationary (time-independent) S2D comparison technique for external arrivals. Then, we consider the case when tardiness is time-varying with factors such as morning traffic affecting tardiness. Last, we consider the phenomena of cases that are scheduled to arrive, but do not show up.

Stationary Analysis. For a stationary analysis, we collect the set of measured (externally) arrived tasks from the set of all tasks T_A in the event log:

$$A_e = \{t \in T_A \mid \forall\ t' \in T_A : \xi_a(t) = \xi_a(t') \Rightarrow t' \not\prec_A t\}. \tag{4}$$

Subsequently, we gather the set of sampled tardiness values D_e. To this end, we consider cases, for which there is a scheduled task for the observed actual task, by using function ν (see Section 3.3).

$$D_e = \{\tau_a(t) - \tau_p(\nu(t)) \mid t \in A_e \ \wedge\ \nu(t) \neq \epsilon\}. \tag{5}$$

Based on the sampled tardiness values D_e, we realise an S2D comparison and test whether the null hypothesis that the median of the distribution of tardiness values is 0. This test is done using non-parametric techniques [27], since, in general, the distribution of tardiness values cannot be assumed to have expected values. Instead of testing whether the schedule corresponds to the median of the distribution, however, one can take a less conservative approach, and test a null hypothesis that the scheduled value is between two quantiles (e.g., the 25^{th} and the 75^{th}).

Time-Varying Analysis. In many processes, tardiness of customers can be assumed to

be time dependent, which is not reflected in the stationary approach. Therefore, we now present an approach to compare stochastic processes (instead of a single random variable) to their scheduled counterparts—an approach that is general enough to be used also for assessing conformance of other time-varying dynamics (e.g., resource capacities).

Let $A(k), k \geq 0$ denote the stochastic process that corresponds to the number of external arrivals at time k in the F/J network constructed from the event log ($\mathcal{F}_A$). Comparing it to the planned arrivals (as defined by $\mathcal{F}_P$) imposes two challenges. First, the sample size for such a comparison will typically be small since each of the two processes is observed only once for a particular time point k. Second, we need to know the distribution of $A(k)$ under the null hypothesis, for every k. For the special case of $A(k)$ being a inhomogeneous Poisson process, this distribution is known, and a test statistic can be constructed [28]. However, when aiming at analysis of general arrival processes, large sample sizes are required.

We overcome the two challenges by working under the assumption that any timestamp k, recorded in the schedule or event log, can be mapped to a finite set of times, $k \in H = \{k_1, ..., k_m\}$, which corresponds to all possible arrival times according to schedule. For instance, H can correspond to times of day, which have scheduled start times, e.g., $k_1 =$ 7:00, $k_2 =$ 7:15, etc. Further, we assume that the differences between scheduled and observed arrivals at k_i, $A(k_i) - A_P(k_i)$, have a non-parametric, case independent distribution G_{k_i}. This implies that customers scheduled to arrive at 9:00 have the same tardiness distribution, regardless of their individual constraints. Under this assumption, the problem of sample-sizes is less critical since a large number of customers may be scheduled to arrive at a particular point in time k_i. In addition, the analysis can be traced back to the comparison of distributions of point values.

We denote the median of G_{k_i} by $M_d^{k_i}$. Then, for each possible arrival time k_i, we test m hypotheses (similarly to the stationary case):

$$\begin{aligned} H_{0,k_i} &: M_d^{k_i} = 0 \\ H_{1,k_i} &: otherwise. \end{aligned} \tag{6}$$

The extraction of information in the event log needed for testing the above hypotheses is based on a function $\pi : \mathbb{N}^+ \to H$ that maps positive integer-valued timestamps (e.g., UNIX timestamps) into H (e.g., time-of-day corresponding to appointment times). Then, we define the scheduled arrivals at time k_i as

$$A_e(k_i) = \{t \in T_P \mid \pi(\tau_p(t)) = k_i \ \wedge \forall\ t' \in T_P : \xi_p(t) = \xi_p(t') \Rightarrow t' \not\prec_P t\}. \tag{7}$$

The sampled tardiness values at time k_i, in turn, are defined as

$$D_e(k_i) = \{\tau_p(\nu(t)) - \tau_a(t) \mid \pi(\tau_p(\nu(t))) = k_i \wedge t \in A_e(k_i) \ \wedge \ \nu(t) \neq \epsilon\}. \tag{8}$$

Based on $D_e(k_i)$ the hypothesis testing technique introduced in the stationary case is applied.

No-Shows. The analysis thus far has been concerned with tardiness of customers that were scheduled to arrive, and actually arrived However, a common phenomenon in scheduled process is termed *no-shows*, customers that are scheduled to arrive, but do not show up.

We analyse no-shows scheduled for a server $s \in S_I$ by means of their time-independent proportion η_s^n. It can be estimated by the number of no-shows N_s to server s, divided by the size of externally arrived tasks A_e. The former is obtained as follows:

$$N_s = |\{t \in A_e \mid \forall\, t' \in T_A : \xi_p(t) \neq \xi_a(t') \ \vee \ \exists\, t' \in T_A : \big(\xi_p(t) = \xi_p(t')\big) \Rightarrow \big(\rho_p(t) \neq \rho_a(t')\big)\}|. \tag{9}$$

Similarly, no-shows can be explored in time-dependent scenarios, in which there is a time-dependent proportion of no-shows, $\eta_s^n(k_i)$ for time point k_i. Then, the definition of $\eta_s^n(k_i)$ is based on time-dependent arrivals $A_e(k_i)$, instead of A_e.

Dynamics Conformance: Customer Classes. Processing times are specific for customer classes, so that the conformance of these classes along with their assignment probabilities is a prerequisite for assessing conformance of processing times.

Conformance analysis related to customer classes is based on all resource types $R_I \subseteq R$, for which the server nodes are in S_I, i.e., that are shared by the model constructed based on the event log and the one derived from the schedule. As a consequence, there is a common set C of customer classes constructed for both F/J networks.

To assess the conformance of the assignment probabilities, we first derive the probabilities of the schedule-based model, $Z_p(c)$, for some $c \in C$, as follows. Let $C_S \subseteq C$ be the subset of classes that actually appears in the schedule. For these classes, we assume a that $Z_p(c)$ is the proportion of scheduled activities that corresponds to class $c \in C_S$.

The probability $Z_p(c)$ for some class $c \in C$ is then compared to the (stochastic) probability of the model constructed from the event log. For each customer class $c \in C$, we test the null hypothesis, $H_{0,c} : Z_a(c) = Z_p(c)$. The procedure to test the hypothesis is similar to testing routing conformance and a χ^2 test is used.

Dynamics Conformance: Processing Times. Processing times of server nodes of the F/J networks are assumed stationary (time-independent), and class-dependent. To assess conformance of processing times, we rely on an S2D comparison that is based on a random variable P_c for the processing time of class $c \in C$. The variable is assumed to come from

a general (non-parametric) distribution G_c. Let M_d^c be the median of G_c. Further, let P_c^p be the planned duration for serving customers of class c (a duration of zero time units is assumed in case a task is unplanned).

Then, we test $|C|$ null hypotheses to check whether the distribution of the random duration for the tasks of class c is 'centred' around the planned duration, $H_{0,c} : M_d^c = P_c^p$.

The relevant statistic for hypothesis testing on non-parametric distribution's quantiles can be found in [27]. The sample of processing times per class, which is used to calculate the test statistic per class is given by:

$$P_c^a = \{\delta_a(t) \mid t \in T_A \ \wedge \ \nu(t) = t' \ \wedge \ \psi(t') = c\}. \tag{10}$$

Dynamics Conformance: Resource Capacity. Resource capacity of a server node is the number of resources that provide service at time k and is defined by a deterministic process. To assess the conformance of resource capacities, we employ a D2D comparison of the scheduled and the actual number of resources. The number of scheduled resources for server node $s_r \in S_{R_P}$ of resource type $r \in R$ at time k is defined as follows:

$$R_{s_r,p}(k) = |\{t \in T_P \mid \tau_p(t) + \delta_p(t) \geq k \geq \tau_p(t) \ \wedge \rho_p(t) = r\}|. \tag{11}$$

The number of servers for the same resource type r observed in the event log at time k is defined as:

$$R_{s_r,a}(k) = |\{t \in T_A \mid \tau_p(t) + \delta_p(t) \geq k \geq \tau_p(t) \ \wedge \rho_a(t) = r\}|. \tag{12}$$

Conformance of resource capacities is then assessed as the difference between the two measures, $R_{s_r,p}(k) - R_{r,a}(k)$.

Dynamics Conformance: Service Policies. To assess conformance of service policies, we compare the schedule under the assumption of an earliest-due-date (EDD) policy with the actual routing observed in the F/J network constructed from the event log. The EDD policy assumed for the schedule-based F/J network, denoted by P_s, selects the task with the earliest scheduled timestamp from a a set of tasks $\{t_1, \ldots, t_n\}$ waiting in the respective resource queue at time k:

$$P_s(\{t_1, \ldots, t_n\}, k) = \underset{t' \in \{t_1,\ldots,t_n\}, \tau_p(t') < k}{\operatorname{argmin}} t'. \tag{13}$$

In the F/J network constructed from the event log, in turn, we observe a deterministic policy, denoted by P_a, that models past decisions. Therefore, we rely on a D2D comparison.

We define an indicator $\mathbb{1}_{P(i)}$, which is equal to one if indeed the i-th past decision in P_a corresponds to Equation 13. Then, we define a similarity measure that quantifies the level of compliance to policy P_s:

$$\chi_P = \frac{1}{|n|} \sum_{i=1}^{n} \mathbb{1}_{P(i)}. \tag{14}$$

5.2 Operational Conformance to Schedule

For the two F/J networks $\mathcal{F}_P$ (*planned*) and $\mathcal{F}_A$ (*actual*), operational conformance assesses the 'predictive power' of the schedule, i.e., it measures how well the schedule predicts the actual execution of the process in terms of performance measures. Formally, let $\pi_t(A)$ be a (possibly time-varying and stochastic) performance measure constructed from the event log, and $\pi_t(P)$ be a deterministic realization of the same performance measure derived from the schedule. By $D(\pi_t(A), \pi_t(P))$, we denote the function that measures performance-related deviations between the event log and the schedule.

In the remainder, we consider three categories of performance measures: (1) capacity-related measures (e.g., resource queues), (2) synchronization delays, and (3) schedule-related measures (e.g., internal tardiness). For each category, a plethora of measures may be used, so that we limit the discussion to a single representative measures per category.

Capacity-Related Measures. A classical capacity-related measure in service processes is the queue length, considered based on stationary and/or time-dependent averages, medians, and quantiles. Queue lengths are transformed into resource delays, e.g., via Little's result and its extensions [29, 30]. Queue lengths and delays are, in turn, important indicators to Quality-of-Service for the scheduled process [31].

Neither the schedule nor the event log directly record resource queues and delays, but only the start times and end times of tasks. However, this enables computation of queue lengths as follows. Targeting the lengths of queues with respect to resources (i.e., class independent queues), the queue length at time k, per resource type $r \in R$, the schedule-based measure, $Q_{r,p}(k)$, is estimated as:

$$Q_{r,p}(k) = |\{t \in T_P \mid \rho_p(t) = r \ \wedge \max_{t' \in V(t)} \tau_p(t') + \delta_p(t') < k \wedge k > \tau_p(t)\}|, \tag{15}$$

with $V(t) = \{t' \in T_P \mid \xi(t') = t \ \wedge \ t' \prec_P t\}$ being the set of tasks that precede t and share the case identifier with t. That is, customers that are in queue for resource r at time k are those for which all previous tasks have ended, and the scheduled start time of the next task (scheduled for r) has already elapsed.

The estimate $Q_{r,a}(k)$ of the queue length of the model constructed from the event log is

defined analogously. It is worth to note, though, that computation of $Q_{r,a}(k)$ will take into consideration unscheduled server nodes.

For comparing the scheduled and actual queue lengths as processes, we take a deterministic view, i.e., we assume that there are no periodic effects. Then, a D2D comparison procedure is applied and the squared difference of the two measures, $Q_{r,p}(k)$ and $Q_{r,a}(k)$, are summed up. This approach is consistent with comparing two fluid models, which are deterministic approximations of queueing systems [32].

Synchronisation Delays. If several tasks are to be completed synchronously, we may observe delays in their synchronization. As an example, consider the fork-join construct in Figure 3b that synchronizes the vital signs performed by the Clinical Assistant, and drug production performed by the Pharmacy. A delay in the drug production process can lead to overall delays for patients that are scheduled for infusion.

To assess the conformance of these delays, we define two concurrency relations $||_P$ (for the scheduled F/J network), and $||_A$ (for the model constructed from the event log), over T_P and T_A, respectively. A pair of tasks is in these relations, $(t_1, t_2) \in ||_X, X \in \{A, P\}$, if and only if t_1, t_2 overlap in their duration. Based on these relations, we quantify the deviation of synchronisation delays for a task that has been scheduled ($t \in T_P$) by $D_X(t)$:

$$D_X(t) = \max(0, \max_{t' \in (T_P \setminus \{t\}), t' ||_X t} (\tau_X(t') + \delta_X(t')) - (\tau_X(t) + \delta_X(t))). \tag{16}$$

Note that any reduction in synchronisation delays with respect to schedule is considered to be positive. Hence, the above measure considers only the deviations, where the synchronisation delay is longer than planned.

The aggregated deviation (again, a D2D comparison) based on all tasks $t \in T_P$ is then obtained by summing up the individual synchronisation delays.

Schedule-Related Measures. Another type of deviation is based on performance measures that are schedule-related. One such measure is tardiness for tasks with respect to internal arrival times, or due-dates. Deviations in terms of internal arrival may not be captured by the aforementioned capacity-related measures or synchronization delays since they may stem from resources that are unscheduled.

In our running example (see Tables 1 and 2), the infusion activity for patient 111 imposes a due-date for their arrival to the infusion-nurse station (10:30). The actual arrival for infusion for this patient occurred at 10:47. The delay was caused by the Pharmacy, which is an unscheduled resource station.

The difference of scheduled and actual arrival times of tasks provides a direct measure to assess the tardiness of an individual task. Aggregating these difference for all tasks is

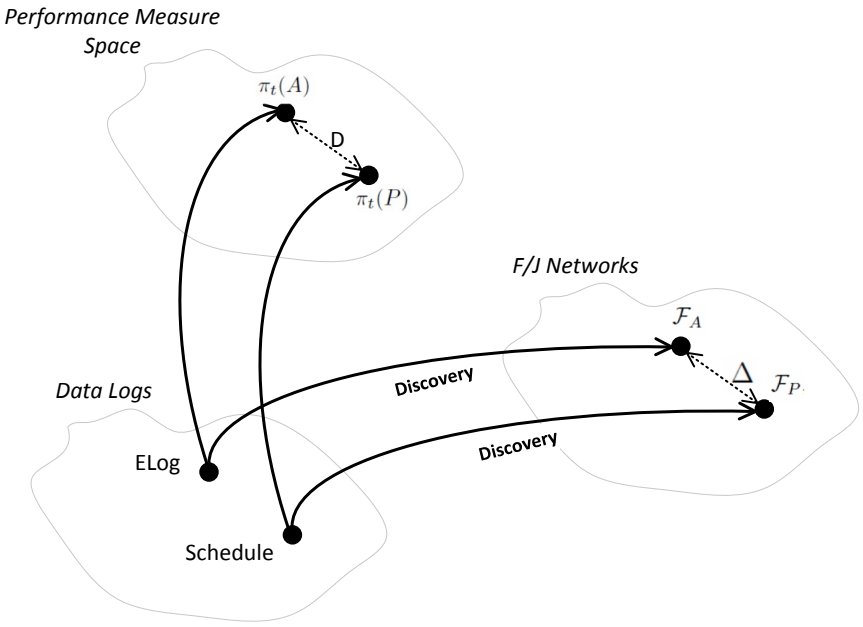


Figure 4: The assumption of continuous conformance

then used as the respective conformance measure.

5.3 The Relation of Conceptual and Operational Conformance

The Continuous Conformance Assumption. Having discussed conformance on the conceptual and operational level individually, we turn to the relation between the two perspectives. In general, both perspectives can be independent, i.e., even in the presence of a conceptual gap between the models for a schedule and the actual process execution, operational predications may be accurate. Similarly, inaccurate predictions may be obtained even if the two models do not differ in terms of their conceptual assumptions.

In many cases, however, the perspectives are indeed related and conceptual conformance can be seen as a prerequisite for operational conformance. We refer to this relation as the assumption of *continuous conformance*.

Applying the assumption of continuous conformance to our setting, we establish a relation between the conceptual gap $\Delta(\mathcal{F}_A, \mathcal{F}_P)$ between the F/J networks discovered from the schedule or event log, respectively, and the difference in operational conformance $D(\pi_t(A), \pi_t(P))$. As illustrated in Figure 4, Δ measures the distance between the F/J networks, D quantifies deviations between the performance measures, while the continuous conformance assumption implies the following:

(1) If the schedule is a good conceptual representative of the actual process execution, corresponding in its assumptions as represented by the paradigm of F/J networks, the schedule is also expected to be an accurate predictor for the actual process execution. This implication advocates scheduling according to real-life assumptions.

(2) If there are deviations in performance measures (i.e., the schedule fails to predict correctly), one may diagnose root-causes for the deviations in inadequate assumptions in schedule. This implication provides us with a heuristic for root-case analysis in the case of operational non-conformance.

An Example for Continuous Conformance. As an example, we assume that $\mathcal{F}_A$ has been discovered from an event log and turns out to be a special case of a F/J network, namely, a single-server queue. For this model, let $A(k)$ be the cumulative arrival process that counts the number of external arrivals up to time t, and let $S(k)$ be the number of departures from service after t units of time. Then, under some restrictions, there exists a well-known mapping tying the processes A and S, i.e., the conceptual assumptions of the F/J network, to the queue length $Q(k)$ at time t. With $\mathbb{1}_{Q(u>0)}$ as the indicator that the queue is not empty at time u, this relation is known as:

$$Q(k) = Q(0) + A(k) - S(\int_0^t \mathbb{1}_{Q(u>0)}\,\mathrm{d}u). \tag{17}$$

The presence of the above mapping justifies the assumption of continuous conformance when assessing the conformance between the model $\mathcal{F}_A$ and another F/J network $\mathcal{F}_P$ that has been discovered from a schedule for the same process. That is, if the cumulative arrival process $A_P(k)$ and departure process $S_P(k)$ defined by $\mathcal{F}_P$ are close to $A(k)$ and $S(k)$, respectively, we are guaranteed to predict the queue lengths accurately.

6 Process Improvement in Fork/Join Networks

Conformance checking (Section 5) detects parts of the process that fail to conform (conceptually or operationally) to a given schedule. In this section, we focus on how to handle lack of conformance by introducing a methodology for process improvement, which combines data-driven analysis via the Fork/Join model, and principles from scheduling research [33].

Provided with the stochastic F/J network, $\mathcal{F}_A$, which corresponds to the underlying process, we target local improvement of service policy, whenever conformance is lacking. We assume that splits and joins have a single layer of resource nodes, a plausible assumption since multiple stages can be aggregated into such a construct [34].

By default, scheduled processes often operate under the Earliest-Due Date first (EDD)

service policy per node, thus 'optimizing' schedule-related performance measures (e.g., non-punctuality). Assuming that all cases are available at the beginning of the scheduling horizon, it is indeed optimal to use the EDD policy (as shown in Section 6.3.1). However, when cases arrive into the system at different times (according to schedule), we show that the EDD policy can be improved to achieve lower tardiness . Moreover, we show that without losing punctuality, the proposed algorithms also improve other performance measures such as flow time. We achieve this by considering synchronization delays in concurrent processing, and by partially re-ordering EDD-ordered cases in a First-Come First-Served order.

As a motivating example we consider two unscheduled resource nodes (clinical assistant and pharmacy, both belong to S_D) from the process depicted in Figure 3b. These unscheduled services cause deviations in punctuality of arrivals to the infusion nurse. The pharmacy is assumed to operate under some service policy (e.g. EDD, FCFS, random order), and we aim at controlling the ordering of patients with the clinical assistant.

In the remainder of the section, we present the optimization setting (Section 6.1) and define the improvement problem (Section 6.2). Then, we present in Section 6.3 two algorithms, showing that they can improve over the EDD policy.

6.1 Optimization Setting

For our concrete optimization task, we focus on a Fork/Join (F/J) construct of $M+1$ parallel resource servers, with $M = 0$ as a special case of a single station. We assume that cases can typically traverse only through one of the splitting branches, while other branches are virtual representatives of that case. In our hospital example, recall Figure 3b, where Clinical Assistant (C.A.) and Pharmacy are performed in parallel. While the patient is physically present in the C.A. station, the pharmacy processes the patient's case without requiring a physical presence. We focus on service policies of the physical branch, where cases are present, and consider the other branches to be uncontrollable (e.g., working according to an EDD policy).

Formally, denote resource server S_0 as controlled by the scheduler and resource servers $S_1, ..., S_M$ controlled exogenously. We aim at improving performance measures, such as tardiness, flow time, and completion time, for the F/J construct in mind.

6.2 Improvement Problem

We now present our optimization problem. We start by presenting input measures (processing times, due dates, release times), and output measures that represent performance (tardiness, flow time), following Pinedo [33].

6.2.1 Input Measures

We introduce three input measures. The first of which is *processing time* ($p_{\xi}^{s,c}$), the time required by a case ξ of class c using a single resource server s (which can be part of a F/J construct). These times are easily obtained by using the δ functions from the two logs.

Next, we separate the scheduled and actual arrival times of a case ξ. The former is called *due date* (d_{ξ}^{s}), representing the point in time at which a case ξ is scheduled to start with a resource server s (possibly a fork station, for F/J constructs). While starting after the due date is allowed, it usually entails a penalty. The latter is called *release date (ready time)* (r_{ξ}^{s}), the time of an actual arrival of a case ξ into service. r_{ξ}^{s} is common for all resource servers $\{S_0, S_1, ..., S_M\}$.

It is worth noting that this notation is easily extended to include more than a single visit of a case with a resource server. We refrain from doing it here for simplictiy sake. Also, whenever the analysis is performed for a single resource server, we can refrain from mentioning s, and stick with p_{ξ}^{c}, d_{ξ}, and r_{ξ}.

6.2.2 Output Measures

We define three performance measures, namely completion time, flow time, and tardiness, all functions of the scheduling decisions.

Completion time (C_{ξ}) of a case ξ is the time at which processing is finished. Let $C_{\xi}^{s,c}$ denote completion times by each resource server, for $s \in \{S_0, S_1, ..., S_M\}$. The completion times of a case ξ of a resource server s is calculated by:

$$C_{(j)}^{s,c} = \max\left(r_{(j)}^{s}, C_{(j-1)}^{s,c}\right) + p_{(j)}^{s,c} \tag{18}$$

where (j) represents the j^{th} processed case and $C_{(0)}^{s,c} \stackrel{def}{=} 0$. Given a case ξ, the completion time of ξ a case at a given resource server s for a service class c is given by:

$$C_{\xi} = \max_{i \in \{0,...,M\}} \left\{C_{\xi}^{S_i}\right\}. \tag{19}$$

It is worth noting that the sequence and processing times on machines $S_1, ..., S_M$ are a priori unknown.

Given completion time, we define next two more measures, using release time and due date, as follows. *Flow time* (F_{ξ}) of case ξ, is the amount of time ξ spends waiting and performing a task:

$$F_{\xi} = C_{\xi} - r_{\xi}. \tag{20}$$

Finally, *tardiness* (T_ξ) of case ξ is the amount of time by which the completion time of ξ exceeds its due date:

$$T_\xi = max(0, C_\xi - d_\xi) \tag{21}$$

6.2.3 Problem Statement

Our goal is to minimize three scheduling criteria, namely maximal tardiness: $T_{\max} = \max_{\xi\in\Xi}\left\{T_\xi\right\}$, maximal flow time: $F_{\max} = \max_{\xi\in\Xi}\left\{F_\xi\right\}$, and the sum of completion times: $\sum_{\xi\in\Xi} C_\xi$. Note that minimizing the sum of completion times is equivalent to minimizing the sum of flow times and sum of waiting times [33].

All three objectives are simple functions of C_ξ. Since the completion times and processing times of resource servers $S_1, ..., S_M$ are a priori unknown, Eq. (19) could be rewritten as follows:

$$\begin{aligned} C_\xi &= \max_{i\in\{0,...,M\}}\left\{C_\xi^{S_i}\right\} \\ &= \max\left\{C_\xi^{S_0}, \max_{i\in\{1,...,M\}}\left\{C_\xi^{S_i}\right\}\right\} \\ &= \max\left\{C_\xi^{S_0}, C_\xi^{S'}\right\}, \end{aligned} \tag{22}$$

where S' is an alternative single machine with $C_\xi^{S'} = \max_{i\in\{1,...,M\}}\left\{C_\xi^{S_i}\right\}$. Thus, without loss of generality, throughout the remainder of this section we refer to a F/J construct with only two resource servers, S and S', with S controlled by the scheduler and S' controlled exogenously.

As before, we distinguish between measures of different resource servers by adding an upper index. Therefore,

$$T_{\max}^s = \max_{\xi\in\Xi}\left\{T_\xi^s\right\} = \max_{\xi\in\Xi}\left\{\max\left\{C_\xi^{s,c} - d_\xi^s, 0\right\}\right\} \tag{23}$$

$$F_{\max}^s = \max_{\xi\in\Xi}\left\{F_\xi^s\right\} = \max_{\xi\in\Xi}\left\{C_\xi^{s,c} - r_\xi^c\right\} \tag{24}$$

6.3 Improvement Algorithms

We are now ready to introduce some properties of EDD service policy in scheduled F/J processes (Section 6.3.1) that naturally lead to two improvement algorithms (sections 6.3.2 and 6.3.3).

6.3.1 EDD properties in Scheduled F/J Networks

We start with optimality of $T_{\max}$.

Theorem 1 *$T_{\max}$ is minimized by minimizing $T^S_{\max}$.*

Proof. In an FJ network the maximal tardiness equals:

$$\begin{aligned} T_{\max} &= \max_{\xi\in\Xi}\{T_\xi\} = \max\left\{C^S_\xi - d_\xi, C^{S'}_\xi - d_\xi, 0\right\} \\ &= \max\left\{\max\left\{C^S_\xi - d_\xi, 0\right\}, \max\left\{C^{S'}_\xi - d_\xi, 0\right\}\right\} \\ &= \max\left\{T^S_{\max}, T^{S'}_{\max}\right\}. \end{aligned}$$

$T^{S'}_{\max}$ is a constant as we have no control over S'; Thus, minimizing $T^S_{\max}$ minimizes $T_{\max}$ ■

Jackson [35] shows the following corollary:

Corollary 1 *EDD minimizes $T_{\max}$ if $\forall\xi : r_\xi = 0$.*

This result points toward the rationale of serving cases EDD, in scheduled processes. However, the following result states that the optimal policy is difficult to obtain, when tasks arrive over time, and are not readily available at the beginning of the scheduling horizon.

Theorem 2 *Minimizing $T_{\max}$ if $\exists\xi \in \Xi | r_\xi \neq 0$ is $\mathcal{NP}$-hard, whether or not the schedule of S' is known.*

Proof. Lenstra et al. [36] showed that minimizes $T_{\max}$ if $\exists\xi \in \Xi | r_\xi \neq 0$ is $\mathcal{NP}$-hard for a single resource server. If the schedule on S' is known, a simple reduction where $\forall\xi : p^{s,c}_\xi = 0$ proves that the problem is $\mathcal{NP}$-hard.

If the schedule of S' is unknown, Theorem 1 proves that in order to minimize $T_{\max}$ the scheduler has to minimize $T^S_{\max}$. According to [36], minimizing $T^S_{\max}$ is $\mathcal{NP}$-hard. ■

Theorem 3 *Sequencing the cases in ascending order of their release times (FCFS order) minimizes $F_{\max}$.*

Proof. Using the proof of Theorem 1 with r_ξ replacing d_ξ, proves that $F_{\max}$ is minimized by minimizing $F^S_{\max}$. Sequencing the tasks in an FCFS order minimizes $F_{\max}$ with a single server (see Pinedo [33]). ■

Theorem 4 *Minimizing $\sum_{\xi\in\Xi} C_\xi$ is $\mathcal{NP}$-hard even if the schedule of server S' is known.*

Proof. In a F/J network

$$F_\xi = \max\left\{C_\xi^S, C_\xi^{S'}\right\} = \max\left\{C_\xi^S - C_\xi^{S'}, 0\right\} + C_\xi^{S'}$$

Thus,

$$\sum_{\xi\in\Xi} F_\xi = \sum_{\xi\in\Xi} \max\left\{C_\xi^S - C_\xi^{S'}, 0\right\} + \sum_{\xi\in\Xi} C_\xi^{S'}$$

$\sum_{\xi\in\Xi} C_\xi^{S'}$ is a constant and minimizing $\sum_{\xi\in\Xi} \max\left\{C_\xi^S - C_\xi^{S'}, 0\right\}$ is equivalent to minimizing $\sum_{\xi\in\Xi} T_\xi$ with a single server where $\forall\xi \in \Xi : C_\xi^{S'} = d_\xi$. Minimizing $\sum_{\xi\in\Xi} T_\xi$ with a single server is $\mathcal{NP}$-hard (see [37]). ■

6.3.2 Combining EDD and FCFS

We next propose an algorithm that combines EDD and FCFS policies to improve over plain EDD policy. To this end, we define a framework for comparison of two sequences with different measures. When a distinction between measures of two competing algorithms is required we denote the measure f produced by a certain algorithm X by $f(X)$ (e.g., f may be maximal tardiness). We say that algorithm X dominates algorithm Y with respect to measure f if for any input $f(X) \leq f(Y)$. We say that X strongly dominates algorithm Y with respect to measure f if in addition $f(X) < f(Y)$ for at least one instance.

The proposed scheduling algorithm strongly dominates the EDD policy with respect to both $T_{\max}$ and $F_{\max}$. The algorithm, denoted A1, start with and iteratively changes an EDD-based sequence. We denote by A1' a temporary sequence held by A1 during its run. A1' changes with the advancement of the algorithm. Finally, A1 is used only when referring to the final sequence of A1.

Line 1 sequences the cases in an EDD order of release time and records $T_{\max}^S(\text{EDD})$. Line 2 re-sequences tasks in an FCFS order, which is the initial sequence denoted by A1'. In lines 7-12, the algorithm transfers cases iteratively as long as there are cases for which the following condition holds:

$$T_{\max}^A(\text{A1'}) \leq T_{\max}^A(\text{EDD}). \tag{25}$$

Γ is the set of positions of all cases whose tardiness exceeds $T_{\max}^S(\text{EDD})$. Ψ is the set of positions of all cases that precede and have a greater due date than the earliest case in Γ. In each iteration of the algorithm we move the case with the maximal due date within the set Ψ right after the first case in Γ, reevaluating Γ after each transfer.

Algorithm 1 From EDD to EDD+FCFS (A1)

1: Order cases in an EDD order and calculate $T^S_{\max}(\text{EDD})$
2: Order cases in an FCFS order
3: Calculate T^S_ξ (A1') for all cases
4: $\Gamma = \left\{\xi : T^S_{[\xi]}(\text{A1'}) > T^S_{\max}(\text{EDD})\right\}$
5: $i = \min_{\xi\in\Gamma}\{j\}$
6: **while** $\Gamma \neq \emptyset$ **do**
7: $\quad \Psi = \left\{j : j < i, d_{(j)} \geq d_{(i)}\right\}$
8: $\quad k = \arg\max_{j\in\Psi}\left\{d_{(j)}\right\}$
9: $\quad$ Transfer case (k) immediately following case (i)
10: $\quad$ Calculate T^S_ξ (A1') for all cases.
11: $\quad \Gamma = \left\{\xi : T^S_{[\xi]}(\text{A1'}) > T^S_{\max}(\text{EDD})\right\}$
12: $\quad i = \min_{\xi\in\Gamma}\{j\}$
13: **end while**

Example 1 *Consider a simple set of five cases:* a, b, c, d *and* e*. Table 3 records for each case, in an EDD order, its processing time, release time, due date, completion time, tardiness, and flow time. Table 4 presents the FCFS order in a similar manner* Γ *now contains one*

ξ	a	b	c	d	e
p^S_ξ	3	5	4	2	1
r_ξ	2	0	1	6	7
d_ξ	5	9	10	13	14
C^S_ξ	5	10	14	16	17
T^S_ξ	0	1	4	3	3
F^S_ξ	3	10	13	10	10

Table 3: EDD order

ξ	b	c	a	d	e
p^S_ξ	5	4	3	2	1
r_ξ	0	1	2	6	7
d_ξ	9	10	5	13	14
C^S_ξ	5	9	12	14	15
T^S_ξ	0	0	7	1	1
F^S_ξ	5	8	10	8	8

Table 4: FCFS order

case, a, which exceeds $T^S_{max}(EDD)$. Both b and c precede a in the FCFS order with smaller due dates (and are therefore in Ψ) and $k = 2$, the second case according to FCFS, which is case c. Therefore, case c is sequenced immediately after case a, as can be seen in Table 5. At this point, no further improvement can be made, Γ is empty and the algorithm halts.

ξ	b	a	c	d	e
p^S_ξ	5	3	4	2	1
r_ξ	0	2	1	6	7
d_ξ	9	5	10	13	14
C^S_ξ	5	8	12	14	15
T^S_ξ	0	3	2	1	1
F^S_ξ	5	6	11	8	8

Table 5: A1 final sequence

We can see that $T^A_{\max}(EDD) = 4 > T^A_{\max}(A1) = 3$ *and* $F^A_{\max}(EDD) = 13 > F^A_{\max}(A1) = 11$.

The run-time complexity of the algorithm is given next.

Theorem 5 *A1 runs in* $O\left(n^3\right)$.

Proof. Case ordering (lines 1 and 2) require $O(n \log n)$ time. In the worst case, each case may be transferred at most $|\Gamma| = O(n)$ times; Hence the maximal number of repetitions of this while loop is $O\left(n^2\right)$. The recalculation of T^S_ξ (A1') (line 10) and the reevaluation of Γ (line 11) require $O(n)$. Hence the overall complexity is $O\left(n^3\right)$. ■

In terms of correctness, we first analyze the relationships between the sets Γ and Ψ in Algorithm A1. We show that $\Gamma \neq \emptyset \rightarrow \Psi \neq \emptyset$, which means that as long as Γ is not empty, the iteration in lines 6-11 can be performed. Lemma 1 below uses the following notations. $\Omega_{(i),(j)}(X)$ denotes the set of cases between the i-th and j-th cases in a given schedule X. $b_{\Omega_{(i),(j)}}(X)$ is the start time of the first case in $\Omega_{(i),(j)}(X)$.

Lemma 1 $C^S_{(j)}(A1') \leq C^S_{(j)}(EDD)$ *for each case whose predecessors all have smaller due dates.*

Proof. We prove this Lemma by contradiction. During the proof we compare two sequences; however, we use the $(\cdot)$ operator to identify cases according to their position in A1'.

Assume that $C^S_{(j)}(\text{A1'}) > C^S_{(j)}(\text{EDD})$ for a certain case (j) and that all the predecessors of (j) have due dates smaller than or equal to its own. Hence, the set of cases preceding case

(j) is a subset of the cases preceding it in the EDD order. Since $C^S_{(j)}\,(\text{A1'}) > C^S_{(j)}\,(\text{EDD})$ there has to be idleness before case (j), as the sum of processing times of the subset cannot exceed the sum of processing times of the superset. Let us consider the largest subset of consecutive case till case (j) (inclusive) with no idle time between them and denote it with $\Omega_{(i),(j)}\,(X)$. That is, (i) is the immediate case after the latest idle time that precedes case (j). The existence of idle time reflects that case (i) started exactly at $r_{(i)}$, which means that it could not have started earlier than it did in the EDD order, as $r_{(i)}$ is a lower bound on case (i)'s starting time. Since $C^S_{(j)}\,(\text{A1'}) > C^S_{(j)}\,(\text{EDD})$ and $b_{\Omega_{(i),(j)}}\,(\text{A1'}) = r_{(i)} \leq b_{\Omega_{(i),(j)}}\,(\text{EDD})$, necessarily $\sum_{\xi\in\Omega_{(i),(j)}(\text{A1'})} p_\xi > \sum_{\xi\in\Omega_{(i),(j)}(\text{EDD})} p_\xi$. Therefore, $\Omega_{(i),(j)}\,(\text{A1'})$ include at least one case that is not in $\Omega_{(i),(j)}$ (EDD), i.e., $\Omega_{(i),(j)}\,(\text{A1'})\setminus\Omega_{(i),(j)}\,(\text{EDD}) \neq \emptyset$. Therefore, there exists a case ξ in $\Omega_{(i),(j)}\,(\text{A1'})$, whose order is $i < k < j$, such that $d_{(k)} < d_{(i)}$.

Let $\xi_{min} = \underset{\xi\in\Omega_{(i),(j)}(\text{A1'})}{\arg\min}\left\{d_\xi\right\}$ be the case with the minimal due date within subset $\Omega_{(i),(j)}\,(\text{A1'})$. Thus, in the EDD order case ξ_{min} is scheduled before case (i), with a due date smaller than $d_{(i)}$). Let (k) denote the position of case ξ_{min} according to A1'. This means that $\Omega_{(i),(j)}\,(\text{A1'}) \subset \Omega_{(k),(j)}\,(\text{EDD})$ and

$$\sum_{\xi\in\Omega_{(i),(j)}(\text{A1'})} p_\xi < \sum_{\xi\in\Omega_{(k),(j)(\text{EDD})}} p_\xi \tag{26}$$

The fact that ξ_{min} is scheduled later than (i) in A1' while $d_{\xi_{min}} < d_{(i)}$ implies that $r_{\xi_{min}} < r_{(i)}$, as A1 starts with an FCFS order and never transfers cases with higher due dates ahead of cases with smaller due dates. This, coupled with Eq. (26) , leads to:

$$\begin{aligned} C^S_{(j)}\,(\text{EDD}) &\geq r_{\xi_{min}} + \sum_{\xi\in\Omega_{(k),(j)}(\text{EDD})} p_\xi \\ &> r_{(i)} + \sum_{\xi\in\Omega_{(i),(j)}(\text{A1'})} p_\xi = C^S_{(j)}\,(\text{A1'}) \end{aligned}$$

which contradicts the assumption that $C^A_{[j]}\,(\text{A1'}) > C^A_{[j]}\,(\text{EDD})$. ■

Example 1 nicely illustrates the proprty in Lemma 1. $C^S_\xi\,(\text{A1'}) < C^S_\xi\,(\text{EDD})$ for all cases ξ but a, the only case whose predecessors' due dates are later than its own. This is true not only for the final schedule, but throughout the algorithm execution. The completion times of the cases that bypassed case c with respect to the EDD order was decreased due to their being pushed ahead of c. The completion times of the cases that are scheduled after c, both in EDD and in A1', was decreased due to the reduction in idle times that usually comes as a property of the FCFS initial order.

To complete our argument, recall that Γ consists of any case ξ that satisfy $T^S_\xi\,(\text{A1'}) >$

$T^S_{\max}(\text{EDD})$. Writing T^A_ξ explicitly, using Eq. (21), we get:

$$\max\left\{C^S_\xi(\text{A1'}) - d_\xi, 0\right\} > \max\left\{C^S_{\max}(\text{EDD}) - d_\xi, 0\right\}$$

Since d_ξ is a constant, unaffected by the schedule, this is possible only if $C^S_\xi(\text{A1'}) > C^S_\xi(\text{EDD})$. Therefore, according to Lemma 1, any case in Γ has at least one predecessor with a higher due date than its own. Thus, we have $\Gamma \neq \emptyset \rightarrow \Psi \neq \emptyset$.

Theorem 6 *A1 strongly dominates EDD with respect to both $T_{\max}$ and $F_{\max}$.*

Proof. We need to show that for any given input $T^S_{\max}(\text{EDD}) \geq T^S_{\max}(\text{A1})$ and $F^S_{\max}(\text{EDD}) \geq F^S_{\max}(\text{A1})$ and that there exists at least one instance with $T^S_{\max}(\text{EDD}) > T^S_{\max}(\text{A1})$ and at least one with $F^S_{\max}(\text{EDD}) > F^S_{\max}(\text{A1})$.

A1 terminates only when $T^A_{\max}(\text{EDD}) \geq T^A_{\max}(\text{A1})$ which establishes domination over the $T_{\max}$ criterion.

We can distinguish between two types of cases in A1, those that are transferred during some stage of the algorithm, denoted J, and those that were not, denoted J'. Since all cases in J were forwarded later in the sequence, all cases in J' are completed no later than their completion time in the initial FCFS sequence. According to Theorem 3, FCFS sequencing guarantees minimal $F^S_{\max}$. Therefore, if $F^S_{\max}$ belongs to a case in J' then obviously $F^S_{\max}(\text{EDD}) \geq F^S_{\max}(\text{A1})$. A case in J is chosen such that it has the maximal due date with respect to all the cases that precede it after the transfer. Thus, according to Lemma 1 $\forall \xi \in J$:

$$\begin{aligned} C^S_\xi(\text{EDD}) &\geq C^S_\xi(\text{A1}) \\ C^S_\xi(\text{EDD}) - r_\xi &\geq C^S_\xi(\text{A1}) - r_\xi \\ F^S_\xi(\text{EDD}) &\geq F^S_\xi(\text{A1}) \end{aligned}$$

which proves that $F^S_{\max}(\text{EDD}) \geq F^S_{\max}(\text{A1})$.

Example 1 shows that there is an instance where $T^S_{\max}(\text{EDD}) > T^S_{\max}(\text{A1})$ and $F^S_{\max}(\text{EDD}) > F^S_{\max}(\text{A1})$. Thus, A1 strongly dominates EDD with respect to both $T_{\max}$ and $F_{\max}$. ■

6.3.3 Scheduling with Completed Cases by S'

The second algorithm we present, A2, starts with any initial sequence (for example EDD, FCFS or A1) and dynamically changes the sequence according to the set of cases completed by server S' in the synchronizing queue. The algorithm changes the sequence as long as

domination over the original sequence is maintained with respect to all three criteria: $T_{\max}$, $F_{\max}$, and $\sum_{\xi\in\Xi} C_\xi$, simultaneously.

At the completion of a service by S, cases that have already completed their service by S' are reevaluated by the algorithm as the optional "next in line" as long as their being pushed forward to the head of the line does not increase any of the three criteria at hand.

If a case is pushed forward the remainder of the sequence remains unchanged at least until the next decision point. Consider a subset of consecutive jobs in the current sequence, $\Omega_{(i)(j)}$, where (i) is the next case in line and (j) is considered as a candidate to be pushed forward. Let us denote the original schedule by INIT, and a run-time sequence (when we make the-reordering decision) by TMP. A change in sequence would occur if the following constraints are upheld:

$$\forall \xi \in \Omega_{(i)(j-1)} : T_\xi^S\,(\text{TMP}) + p_{(j)} \leq T_{\max}^S\,(\text{INIT})\,; \tag{27}$$

$$\forall \xi \in \Omega_{(i)(j-1)} : F_\xi^S\,(\text{TMP}) + p_{(j)} \leq F_{\max}^S\,(\text{INIT})\,; \tag{28}$$

$$\sum_{\xi\in\Omega_{(i)(j-1)}} p_\xi \geq p_{(j)}\,(j-i)\,. \tag{29}$$

The pseudocode of Algorithm A2 uses the following notations. $\Upsilon^s\,(t)$ is the set of cases that completed their service s (either S or S') at time t and are now in the synchronizing queue. A case ξ is added to $\Upsilon^{S'}\,(t)$ if $C_\xi^{S'} \leq t$ and is subtracted from $\Upsilon^{S'}\,(t)$ if $C_\xi^S \leq t$ and vice versa. Note that $\forall t : \Upsilon^S\,(t) \cap \Upsilon^{S'}\,(t) = \emptyset$ since if both arrive at the synchronization queue they are immediately removed from it. Let $\Phi\,(t_1, t_2)$ be an exogenous function that returns the set of cases that completed their process on machine S' during $(t_1, t_2]$. $\sigma = \{(i+1), \ldots, (n)\}$ denotes the planned sequence for the remaining $(n-i)$ cases assuming i cases have already been served by S. σ is initially set according to INIT but may change during the run of A2. We denote by $\sigma\,(i)$ the i^{th} case in the ordered sequence σ at the time function $\sigma\,(i)$ is called. Accordingly, $\sigma\,(1)$ denotes the "next in line" scheduled job.

We shall illustrate Algorithm A2 with the following simple example.

Example 2 *To illustrate the use of A2 we pick up where we left off with Example 1. That is, we use A2 on an initial sequence given by A1. In our example the completion times of S' (revealed online and not known apriori) are as follows:*

ξ	b	a	c	d	e
$C_\xi^{S'}$	6	4	13	12	7

Table 6: completion times of S'

At $t = 8$, the first two cases b and a were served. At that time (computed in lines 33-37)

$$\Upsilon^S(8) = \emptyset; \Upsilon^{S'}(8) = \{e\}; \sigma = \{c, d, e\}.$$

That is, case e is waiting in the synchronization queue for its completion on machine S. Currently, cases c and d are scheduled ahead of case e for S. We consider pushing e ahead of c and d. However, the second condition in line 17 is unsatisfied and we continue with the initial schedule once again ending up at $t = 12$ with:

$$\Upsilon^S(12) = \{c\}; \Upsilon^{S'}(12) = \{d, e\}; \sigma = \{d, e\}.$$

We now illustrate the execution of the while loop (lines 5-38) for $t = 12$. In line 6 we set $j = 2$; $s = 1; \Delta_{\min} = 0$. $\sigma(2) = e$, it satisfies both conditions of line 7: $\sigma(2) = e \in \Upsilon^{S'}(12) \cap \sigma \backslash \sigma(1) = \{e\}$ and $1 = p_e < p_d = 2$ Therefore, we set $k = 1$ (line 8) and move to line 12, verify that $k \leq j$ and set $C_d^S = \max\{12, 6\} + 1 + 2 = 15$ (line 13). Both conditions of line 17: $C_d^S - d_d = 2 \leq T_{\max}^S(INIT) = 3$ and $C_d^S - r_d = 9 \leq F_{\max}^S(INIT) = 11$ hold. Being the last element of σ we can now move to line 23. We set $\Delta(e) = p_e(5-4) - \sum_{i=1}^{1} p_{\sigma(i)} = -1.$, which is better than the previous value of Δ_{min} and therefore, in lines 25-26 we set $\Delta_{\min} = \Delta(j) = -1$ and $s = 2$. As a result, S provides service to e instead of d. By swapping cases d and e, A2 decreases the final sum of completion times of INIT by at least $\Delta_{\min} = -1$. Table 7 sums up the final sequence of A2 as well as all the relevant measures associated with it.

ξ	b	a	c	e	d
p_ξ^S	5	3	4	1	2
r_ξ	0	2	1	7	6
d_ξ	9	5	10	14	13
C_ξ^S	5	8	12	13	15
$C_\xi^{S'}$	6	4	13	7	12
C_ξ	6	8	13	13	15
T_ξ	0	3	3	0	2
F_ξ	6	6	12	6	9

Table 7: Outcome of Algorithm A2

Theorem 7 *A2 dominates INIT with respect to $T_{\max}$, $F_{\max}$ and $\sum_{\xi \in \Xi} C_\xi$.*

Proof. Conditions (27) and (28) (tested in line 17 of the algorithm) quite straightforwardly guarantee that the delay caused to any of the tasks in $\Omega_{(i)(j-1)}$ does not allow their

tardiness and flow time to exceed $T^S_{\max}(\text{INIT})$ and $F^S_{\max}(\text{INIT})$, respectively.

A case (j) will be pushed forward, ahead of cases $(i),\ldots,(j-1)$ only (but not necessarily) if Condition (29) is satisfied. Since (j) is assumed to have also completed its service with S', The sum of processing times of the cases it bypassed, $\sum_{\xi\in\Omega_{(i)(j-1)}} p_\xi$, represents a lower bound to the actual decrease in its completion time. On the other hand, $p_{(j)}(j-i)\leq 0$ represents a upper bound on the increase in the completion times of all the cases that were originally ahead of (j) in the queue for S alone: $\sum_{s=i}^{j-1} C^S_{(s)}$ (both bounds are tight if there is no idle time between servicing (i) and (j)). This increase is an upper bound on the overall completion time $\sum_{s=i}^{j-1} C_{(s)}$. Hence, a change in sequence occurs only if a decrease in $\sum_{\xi\in\Xi} C_\xi$ is guaranteed. Thus, A2 dominates INIT with respect to $\sum_{\xi\in\Xi} C_\xi$ as well. ■

Theorem 8 *Assuming that the sequence INIT is given and $\Phi(t_1,t_2)$ runs in $O\left(n^2\right)$, A2 runs in $O\left(n^3\right)$.*

Proof. Lines 2-4 require $O(n)$ time. The while loop repeats $O(n)$ times. Within this loop, the indicws j (outer) and k (inner) each takes $O(n)$ times. Note that the summation in lines 13 can be done incrementally and does not add to the complexity. Line 23 runs in $O(n)$ time. Line 35 depends on the complexity of $\Phi(t_1,t_2)$, assumed to be $O\left(n^2\right)$. Thus, A2 runs in $O\left(n^3\right)$. ■

It is worth noting that A2 may be easily adopted to dominate only a subset of the measures $(T_{\max},F_{\max})$ by disregarding the respective condition in lines 17-22. This relaxation of the constraints would probably allow better performance on the $\sum_\xi \in \Xi C_\xi$ measure. Both A1 and A2 are heuristic solutions, guarantee not to do worse than the ordering given as input, yet with no guarantees on how far they are from the optimal solution.

As a final note, we highlight the fact that the analysis is localized to a specific part of the F/J network, and relating to a single server only. While the localization of the analysis fit well the optimizations of conformance failures, it may also miss out on possible inter-relationships among different parts of the network. We leave the analysis of multi servers and golbal network analysis to future research.

7 Evaluation

This section presents an empirical evaluation of the proposed methods of algorithms using data of the Dana-Farber Cancer Institute, see Section 2. The experiments consist of two parts, namely conformance checking and process improvement.Section 7.1 presents an analysis of the conformance of the service process for on-treatment patients (OTP) to its schedule. This analysis highlights operational deviations and links them to root causes in terms of

conceptual conformance issues. In particular, we identify a synchronisation delay in the process that is not expected according to the schedule and which can be traced back to the service policy implemented by the pharmacy server node.

Section 7.2 illustrates the benefits of the process improvement algorithms. By aiming at an optimisation of the service policy of the clinical assistance station we demonstrate that the median tardiness and median processing delay can be improved by more than 20%.

By evaluating service policies in both parts, we demonstrate how conformance checking and process improvement can be linked through data-driven queueing networks. We conclude the section with a discussion of our approach with its limitations, and provide a view on overcoming these limitations with the results of the experiments (Section 7.3).

7.1 Conformance Checking

Below, we first describe the dataset and experimental setup for the conformance checking evaluation. Then, we turn to the results in terms of operational and conceptual conformance.

Datasets and Experimental Setup. Our experiments combine three data sources from the Dana-Farber Cancer Institute: an *appointment schedule*, an *RTLS log* recording movements of badges (patients and service providers), and a *pharmacy log* that records checkpoints in the medication-production process. As such, the event log as introduced in Section 3 is actually split up into the RTLS log and the pharmacy log. The resolution of the RTLS is around 3 seconds, depending on the amount of movement of a badge. The pharmacy log is also rather accurate, since process checkpoints are prerequisites to move to the next stage of production. From the pharmacy log, we only extracted the start and end events for the production process, since the pharmacy is considered as a separate server. The experiments involve three weekdays, April 14-16, 2014, which are days of 'regular' operation (approximately 600 OTP patients) as was verified with local contacts.

Using this dataset, we first discovered the F/J networks from the event logs and the schedule. as detailed in Section 4.2. Figure 3a illustrates the structure and deterministic routing obtained from the schedule, which is plain sequence of resource servers. Figure 3b, in turn, presents the structure and routing as found in the event log including additional resource nodes (Clinical Assistant and Pharmacy) as well as a probabilistic routing (e.g., activity conducted by the infusion nurse is skipped with probability 0.1).

We then assessed the operational conformance to identify deviations between the schedule-based model and the model constructed from the event log. Here, we focus on findings regarding the processing delay, specifically the synchronisation delay of a patient that is scheduled to enter infusion. Such a patient has to wait for two concurrent activities, namely

pre-infusion vital signs (vitals) and medication production. According to the schedule, there is no delay between the end of vitals and the beginning of infusion.

Since this analysis reveals deviations in the scheduled and actual delay, we then turn to conceptual conformance checking to identify potential root causes for this deviation. Specifically, we explore the conformance of the server dynamics for the resource node at which the deviation has been observed.

We implemented our experiments in two software frameworks, SEEStat and SEEGraph. Both tools have been developed in the Service Enterprise Engineering lab,[2] and enable, respectively, statistical and graphical analysis of large operational datasets. In particular, they enable the creation of new procedures for server dynamics (SEEStat), and for the discovery of structure and routing in queueing networks (SEEGraph).

Operational Conformance: Processing Delays. Figure 5 presents the actual distribution of time between vitals and the beginning of infusion, based on the RTLS data. We observe that, indeed, a large portion of patients go into infusion within a minute from vitals. However, the distribution presents a long tail of patients, who wait for the next step with an average delay of 23 minutes. The hypothesis that the synchronization delay conforms to a median value of 0 is rejected for any significance level. In many occasions, one can observe in the RTLS data that patients wait, while infusion nurses are available for service. For most patients, this is due to synchronization delays since they wait for their medications.

This points toward synchronization delays between the vitals activity and the pharmacy. According to schedule, the central pharmacy is planned to deliver the medication in synchronization with vitals (within 30 minutes). The operational insight of long synchronization delays, however, hints at a conceptual issue regarding the just-in-time arrival of the medication.

Conceptual Conformance: Service Times and Policy. Taking the structure and routing of the F/J network illustrated in Figure 3b, we assume that the fork is zero-delay and that the pharmacy is notified once the patient is ready for infusion. Therefore, we assume that the arrival times do not deviate and diagnose the two remaining dynamics: production time and service policy.

Figure 6 shows the distribution of production times (for April 2014), and the fitted Dagum distribution.[3] Here, we observe that the stochastic model shares a first moment with the planned duration: both are 30 minutes on average. This is an S2D comparison, and the result of hypothesis testing for the median of processing time being 30 minutes is not rejected with a significance level of 5%. Therefore, the duration of the service does not

[2] `http://ie.technion.ac.il/labs/serveng/`

[3] `https://en.wikipedia.org/wiki/Dagum_distribution`

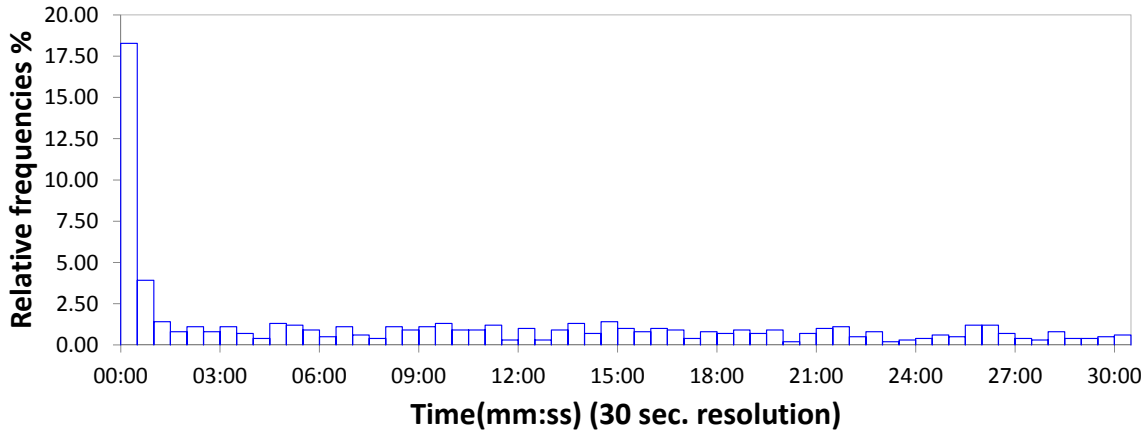


Figure 5: Waiting time for Infusion (after vitals); Sample size = 996, Mean = 25min, Stdev = 29min

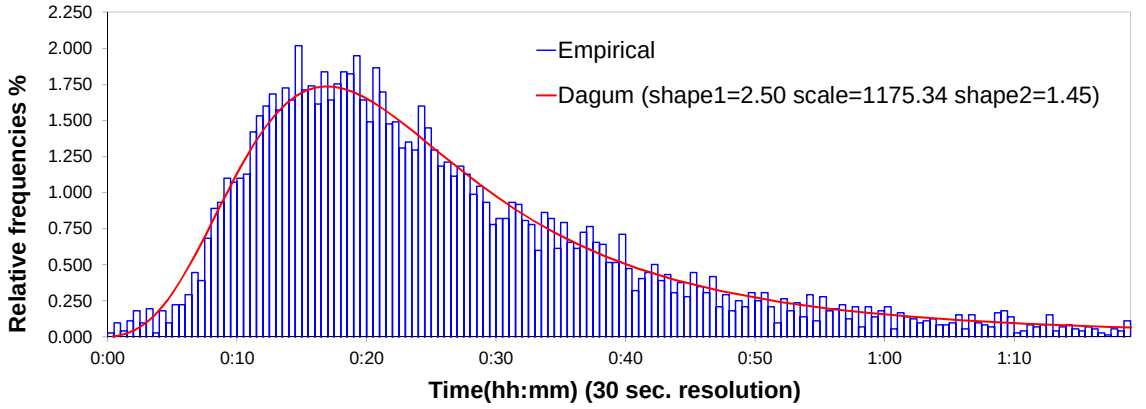


Figure 6: Medication production time; Sample size = 7187, Mean = 30min, Stdev = 24min.

explain the operational deviation identified above.

Turning to the service policy, we realised a D2D comparison and adopted the similarity measure that we defined in Section 5.1 for comparing policies. We focus on the time until the first drug has been prepared. Although patients often require more than one drug, the first medication is the one that is needed for the process to progress. Using the method proposed in Section 5.1, we estimated the expected indicators for three policies: (1) Earliest-Due-Date (EDD) first, which corresponds to the policy defined in the schedule, (2) First-Come First-Served (FCFS), which produces according to the order of prescription arrivals and (3) Shortest-Processing-Time (SPT) first, which implies that priority will be given to patients with shorter infusion durations.

Figure 7 presents the estimated proportion of compliance to policy, as a function of

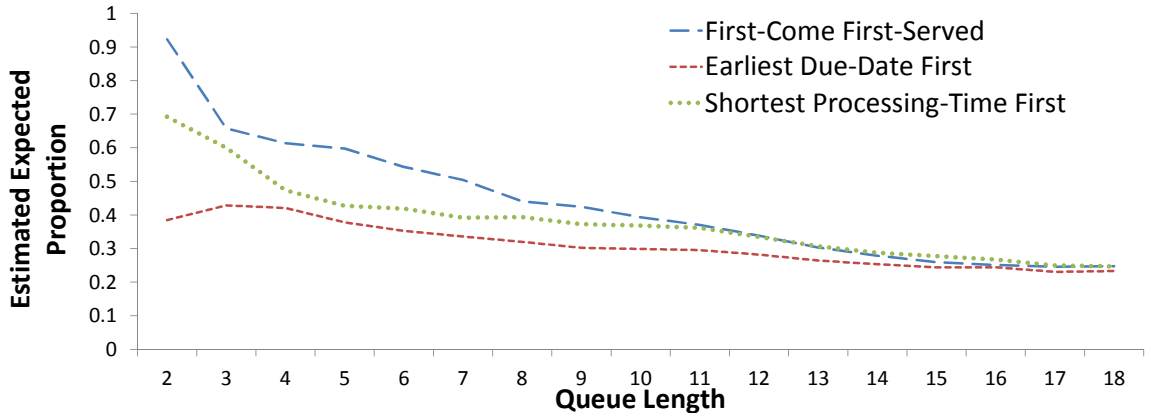


Figure 7: Policy comparison for the pharmacy resource

the number of medication orders in queue. To see an effect of selection based on a policy, the comparison starts with a queue of size two. We observe that as the queue grows, the decision on the next task to enter service becomes more random. However, for short queues, the selection policy tends towards FCFS, instead of EDD as assumed in the schedule. The deviation between the two policies, planned according to schedule and actually observed in the event log, can be seen as a cause of the synchronization delays identified above.

7.2 Process Improvement

Both Algorithm 1 and Algorithm 2 guarantee not to worsen performance measures, yet it is unclear whether a significant improvement can be gained. Moreover, the guarantee is only for certain measures such as maximal tardiness and flow time, while no guarantees are provided for other important measures, such as median tardiness and flow time. We set to test the projected improvement for both types of measures (guaranteed and non-guaranteed) by considering DFCI data again.

Specifically, we return to the pharmacy-patient flow interaction, which involves two resource servers, see Figure 2b. Given that pharmacy is sequencing its jobs in an uncontrolled order (as we demonstrated above), we aim at improving the service policy of the clinical assistant station. In the remainder, we provide the dataset and setup for the experiments. Then, we go over the main results, and demonstrate the improvement.

Datasets and Experimental Setup. The data is taken from 147 working days of the Dana-Farber Cancer Institute in 2014. We consider only January-August, for which we have both patient and pharmacy data. As input to the case sequencing problem, we consider the actual patients that arrived along with their actual processing times, due dates and release

times.

We sequence the processing order in the clinical assistant station according to four possible service policies: (1) the actual ordering of patients as it was observed in the data; (2) EDD ordering (according to the scheduled start of infusion step); (3) the ordering defined by Algorithm 1 with EDD as the initial sequence; and (4) the ordering defined by Algorithm 2 with the output of Algorithm 1 as the initial sequence. The first sequence serves as a baseline for the last three sequences.

Once patients are sequenced, a *deterministic* trace-based simulation runs to calculate completion times, along with all relevant performance measures, for all patients.

For Algorithm 1 the non-decrease in performance with respect to EDD is guaranteed for maximal tardiness and flow time, while for Algorithm 2 the guarantee is for any initial sequencing with respect to the sum of completion times (correlated with flow time), maximal tardiness, and maximal flow time. Therefore, we first compare the outcomes of the four sequences with respect to the three guaranteed performance measures, namely sum of completion times, maximal tardiness, and maximal flow. In addition, we add two measures that better represent the experience of an 'average' patient, median tardiness and median flow time, for which the algorithms give no guarantees.

We start by comparing performance measures for all arriving patients. However, the theoretical guarantees that we show in Section 6 are for single-server stations. Therefore, to show a more realistic setting, we move to comparing performance at the resolution of specific classes of patients. To this end, we cluster the patients according to their individual diagnoses. Since, in practice, clinical assistants are dedicated per diagnosis, we get a scenario that is closer to the single-server assumption of our analysis.

Results. Figure 8 presents improvement over the baseline, in percentage, with negative values corresponding to improvement in performance. For example, the value of -20.66% for F_{median} corresponds to improvement of 20.66% with respect to the simulated median flow when actual sequencing is applied.

Not surprisingly, we observe that the three guaranteed measures, for both Algorithm 1 and Algorithm 2, improve over EDD (schedule-driven policy). Furthermore, we observe that for median flow time and median tardiness, there is a large improvement for Algorithm 2. Overall, Algorithm 2 improves on Algorithm 1 in all performance measures. This is due to the consideration of the synchronization queue, which does not exist in Algorithm 1.

Moving from all patients to specific diagnoses, Figure 9 shows a similar analysis for hematology malignancies patients. These patients are treated in a separate disease center, and have dedicated clinical assistants. We observe that the behavior of the results is similar to the overall population, yet with bigger improvement. We believe that this improvement

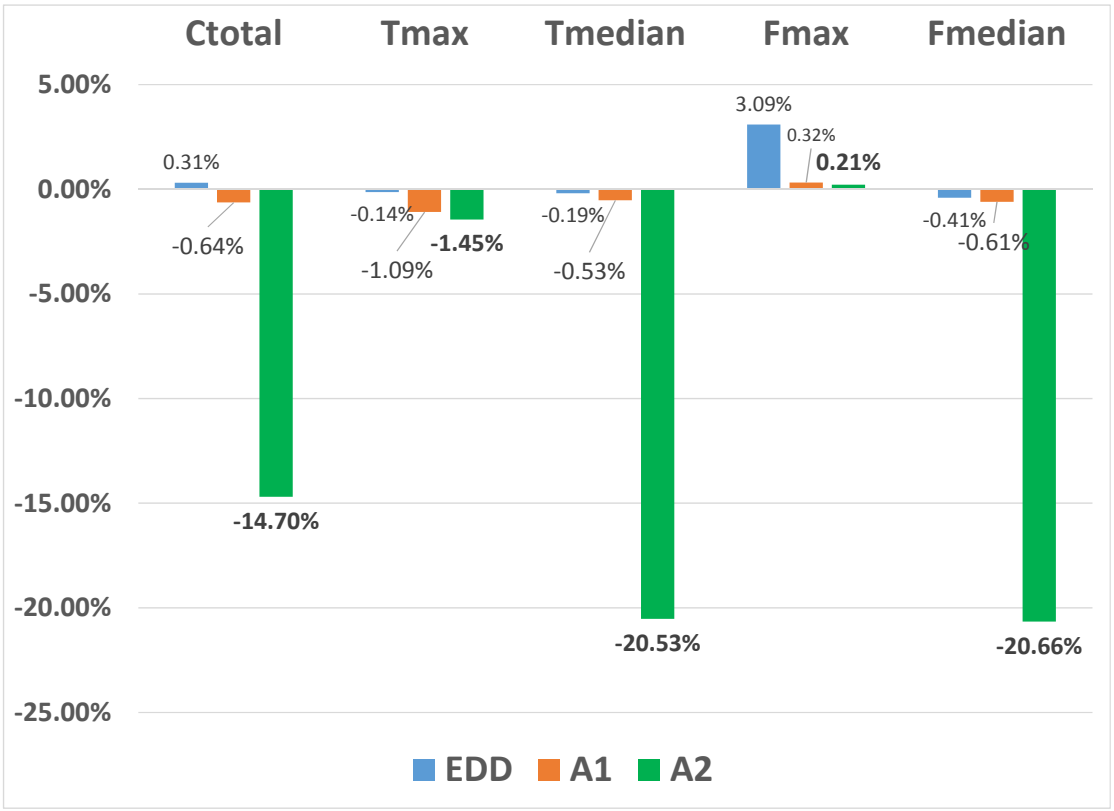


Figure 8: Process improvement results: all patients.

stems from higher conceptual conformance of the single-server assumption for these patients.

Finally, Figure 10 presents the improvement in sum of completion times as a day-of-week function. Wednesdays are known to be the most loaded days in Dana-Farber Cancer Institute (Figure 11). We observe that the highest improvement rate is obtained for the more loaded days. This result puts forward the importance of correct ordering, especially on heavily loaded days.

7.3 Discussion and Limitations

We now discuss of the main results of our evaluation with respect to both conformance checking and process improvement.

Conformance to Schedule. Our approach to assessing conformance to schedule was demonstrated to be useful in detecting bottlenecks and finding their corresponding root-causes. Specifically, we instantiated our methods on real-life data, analyzed the pharmacy

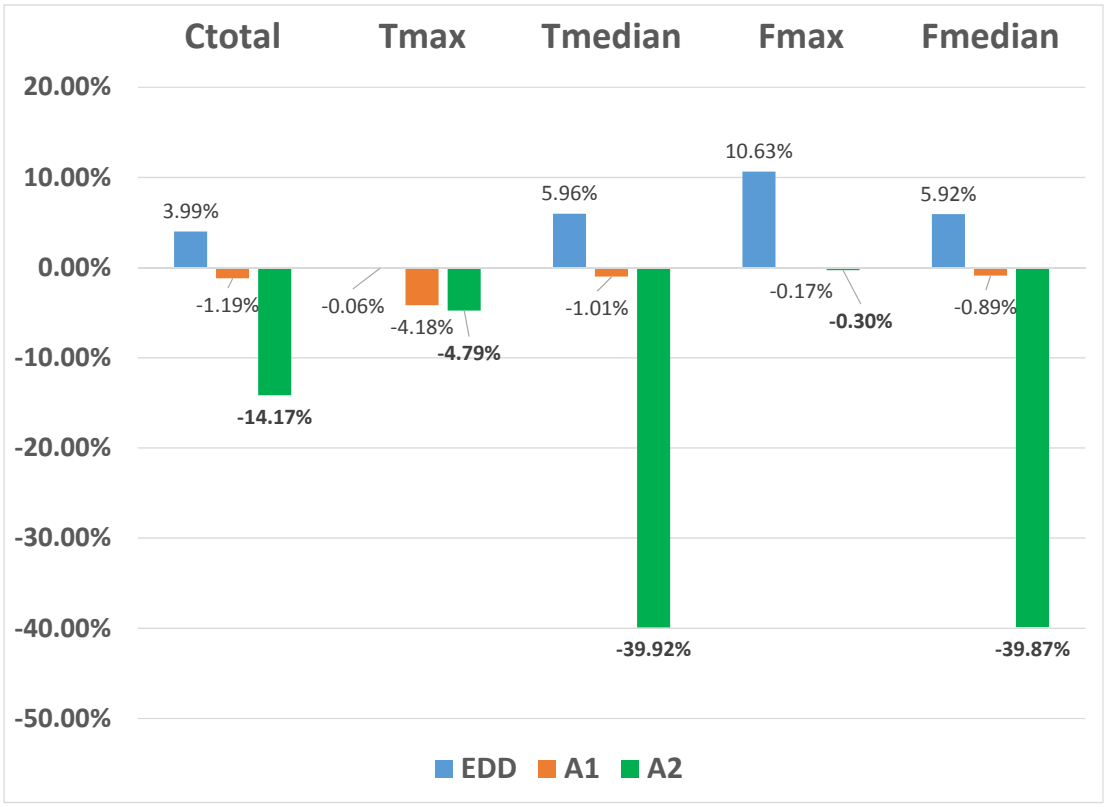


Figure 9: Process improvement results: hematology patients.

station, and found its service policy to be the cause for the bottleneck in patient flow. However, in order to create a full multi-dimensional comparison between a schedule and a process execution we need to combine the deviations. This requirement gives rise to two limitations of the approach: (1) we perform both D2D and S2D comparisons, thus giving heterogeneous answers to conformance questions (p-values vs. similarity measures), and (2) when testing multiple hypotheses one needs to be careful with assessing the correctness of the overall rejection rate [38].

To handle the first limitation, one may consider the resulting p-value (in S2D comparisons) to be a similarity measure, and come up with a unified single measure to quantify the distance between planned and actual. The second limitation can be solved by employing techniques from the statistical field of multiple comparisons [38], which provides appropriate tools to simultaneously testing multiple hypotheses.

Process Improvement. Our experiments demonstrate about a 20% increase in process performance, using performance measures such T_{median} and F_{median}. Also, in loaded days

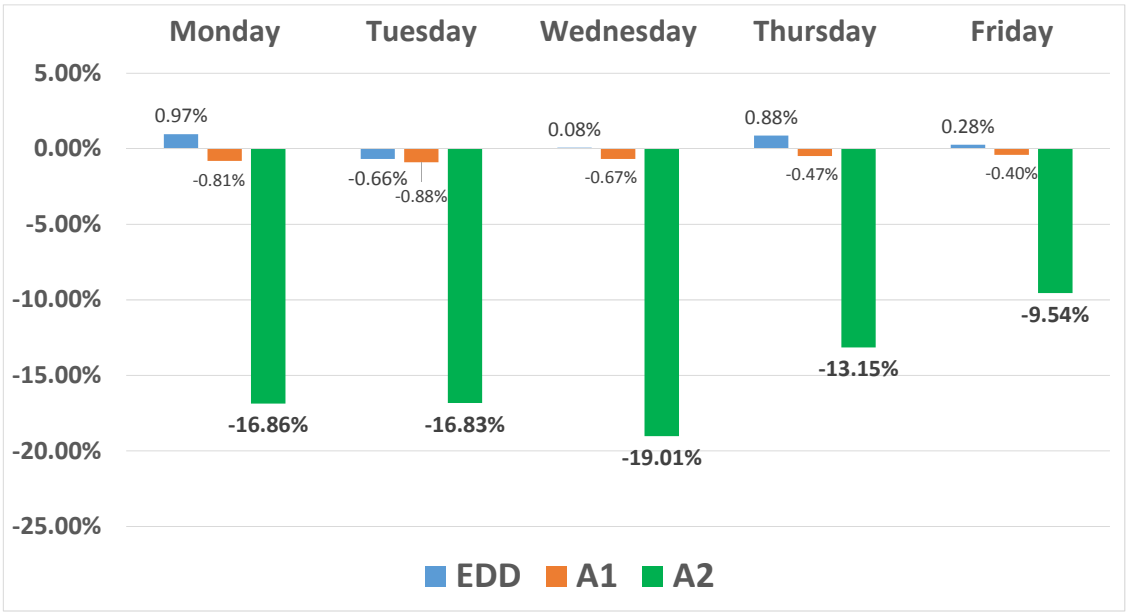


Figure 10: Process improvement results for sum of completions: weekdays.

with more cases in the system the algorithms perform even better. These are definitely encouraging results. However, the proposed techniques suffer from several conceptual limitations. First, the techniques are applied independently for each Fork/Join construct. This clearly implies an independence assumption between stations in the network, which is not always the case. Second, our approach assumes that we can only control a single branch of the Fork/Join construct, therefore simplifying the optimisation problem. Third, it is assumed that stations are of single-server type, which is an approximation to a many-server reality that one encounters in typical service processes. On the one hand, assuming a fast single-server as an approximation to many-servers was found accurate for several special cases [39]. On the other hand, well-established results show that different analysis techniques are required to capture the operational behavior of many-server stations [40].

To handle these conceptual limitation, one needs preliminary analysis of the data, to test whether the assumptions of independence, controllability, and server capacity, hold. This brings back the need to assess conceptual conformance. An encouraging indicator to the need of such preliminary analysis can be found in Figure 9, where conceptual conformance to these assumptions yields double the improvement in performance.

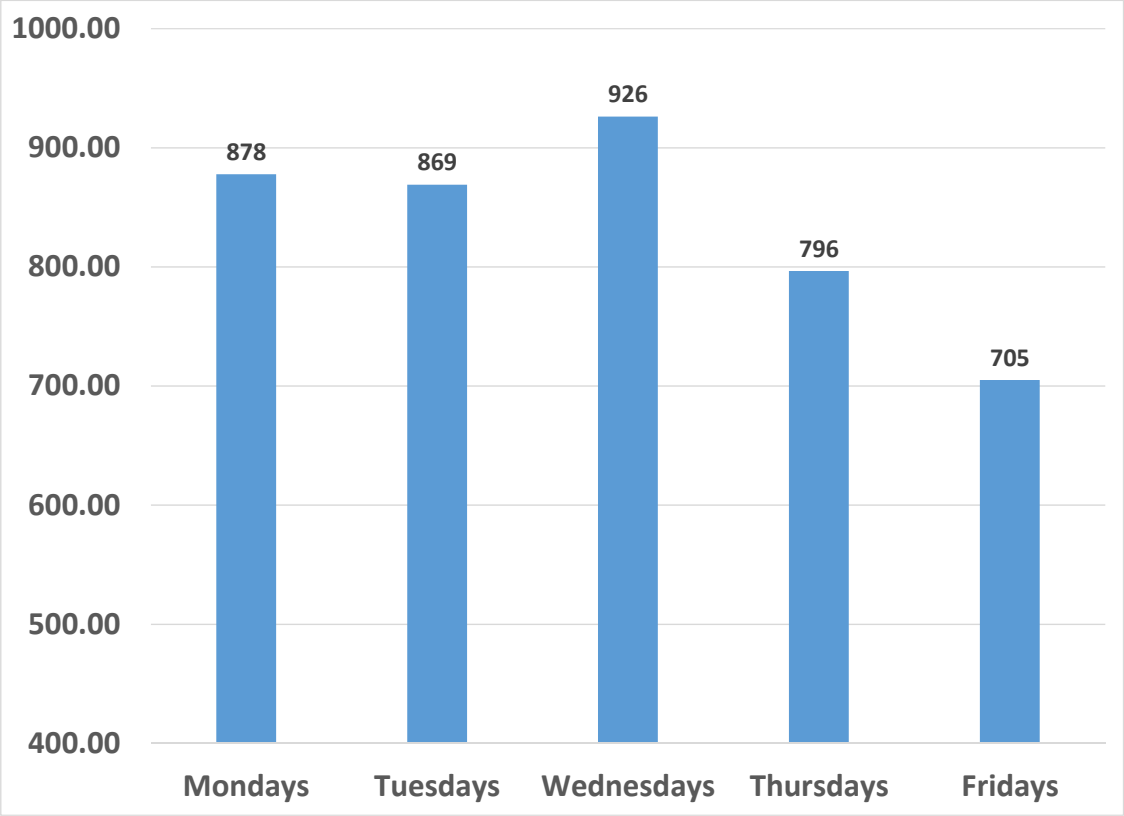


Figure 11: Number of patients per day: weekdays average.

8 Related Work

Recently, there has been an increased interest in scheduled service processes, especially in the health sector. Outpatient clinics that operate as a scheduled multi-stage process, are of particular interest, due to their pervasiveness and growth over the past years [3]. Performance questions for scheduled multi-stage processes relate to bottleneck identification, dynamic resource allocation, and optimal control (decision making). Our solution to these questions is based on three central elements, namely process modeling, assessing conformance to schedule, and process improvement via principles of scheduling. In the remainder of the section, we discuss related work in these three areas, explaining how our approach advances the state-of-art.

Process Modeling. Traditionally, operational process analysis is based on modeling methods from Operations Research disciplines, such as Queueing Theory [41]. These methods use hand-made (highly abstract) models of reality, and apply relevant (model-

specific) analysis. The discovered process perspective is typically poor in detail, and is seldom automatically extracted from data. Moreover, validation of the results is typically performed by simulating reality (again a hand-made model), and comparing the outputs of the modeled reality and the simulated reality, neglecting the benefits of data-driven validation, c.f. [42].

The rapidly evolving field of *process mining*, in turn, is driven by data [17]. Models are *discovered* from and validated against event data from recorded process executions, see [43]. Mined models are used as the basis for prediction [44, 45], simulation [46], and resource-behavior analysis [47, 48]. However, much work in this field focuses on the *control-flow perspective*, i.e. the order of activities and their corresponding resources, times and decisions [17, Ch. 8], so that the created models cannot benefit from the analysis techniques developed in Operations Research.

In earlier work, therefore, we argued for an explicit representation of the *queueing perspective* and demonstrated its value for several real-world processes [20, 22]. However, the existing techniques all considered the simplistic setting of a single-station system, whereas, this paper addressed the more complex scenario of service processes that are scheduled and have a multi-stage structure that involves resource synchronization.

Conformance to Schedule. One of the main questions in scheduled processes is that of conformance of the actual process execution to the plan. Techniques for conformance checking in process mining primarily focus on behavioral conformance, see [49, 50, 51, 52]. A few works also addressed time and resource conformance of discovered models [53, 54], where the replay method, as in [55], is used to quantify deviations in performance measures. In these approaches, conceptual conformance (model assumptions) is mixed with operational conformance (resulting performance measures). This paper argues for a clear separation between operational and conceptual conformance, and introduces a methodology for assessing the operational and conceptual validity of Fork/Join networks. Moreover, we link the two conformance types together via continuous conformance.

Another related line of work is concerned with business process deviance [56]. Here, two business processes are compared either by comparing a normative model to a log, or by comparing two logs (log Delta analysis) [57]. State-of-art literature in business process deviance is mainly concerned with the control-flow perspective, and other perspectives (i.e., time, resources) are neglected. Our approach for conformance to schedule covers all operational perspectives when comparing two data logs.

Typically, process mining techniques for conformance checking are concerned with deterministic to deterministic comparisons only, since the underlying models are assumed to be deterministic in nature. In this work, we consider some of the process elements, such

as processing times and arrival rates to be stochastic. Further, we provide techniques for stochastic to deterministic comparisons.

Process Improvement with Scheduling. In deterministic scheduling, problem settings with due-dates and non-zero release times have been extensively studied [33]. Yet, we are not aware of literature that covers the Fork/Join setting as it is presented in Section 6.

Fork/Join networks are stochastic models that naturally arise when modeling service processes. Therefore, most of the existing literature on scheduling F/J networks assumes stochastic processing times [8]. The theoretical results in these works are typically limited to approximations, and to restrictive assumptions (e.g., single-station models, heavy-traffic assumptions). Our work uses a deterministic setting and provides two novel algorithms for improving performance.

Both stochastic and. deterministic processing times have merit in different settings. Therefore, in processes with high variability in activity durations, one should consider the stochastic setting, while for processes with low variability the deterministic assumption can yield good approximation.

9 Conclusion

In this work, we provided methods for conformance checking and performance improvement of scheduled multi-stage service processes, as they are observed in such domains as healthcare and transportation. To assess the conformance of a schedule of a process to its actual execution, we presented an approach based on process discovery and statistical analysis of Fork/Join networks. The discovered Fork/Join network was then facilitated to improve the underlying scheduled process via techniques of mathematical scheduling. Specifically, we developed theoretical results that guarantee a non-decreasing performance measures, such as tardiness and flow time, when ordering cases for concurrent processing.

We evaluated the approach with real-world data from an outpatient clinic in two steps. First, we showcased how our approach for conformance checking leads to the identification of operational bottlenecks and supports the analysis of the root-causes of these bottlenecks. Second, we demonstrated the value of our process improvement approach by simulating alternative scheduling realities that used case orderings as recommended by our algorithms. The experiments resulted in a 20%-40% improvement, with respect to a baseline of the actual ordering of cases.

In future work, we aim at developing a unified measure for quantifying the gap between scheduled and actual execution, with accent on mixing D2D and S2D comparisons, and multiple hypothesis testing. We plan to facilitate the quantified gap between scheduled and

actual process executions for improving prediction of outcomes in the scheduled process (e.g., performance measures, patient behavior). Furthermore, we target the extension of the proposed conformance checking techniques, to allow comparison of the resource perspective of normative models to execution data. Last but not least, we aim at lifting our results for process improvement to stochastic processing times, many-server stations with less restrictions on process controllability.

Acknowledgment. We are grateful to the SEELab members, Dr. Valery Trofimov, Igor Gavako and especially Ella Nadjharov, for their help with the statistical analysis. We also thank Kristen Camuso, from Dana-Faber Cancer Institute for the insightful data discussions.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Springer, 2013. 2

[2] M. S. Daskin, Service Science, Wiley. com, 2011. 2

[3] C. M. Froehle, M. J. Magazine, Improving scheduling and flow in complex outpatient clinics, in: Handbook of Healthcare Operations Management, Springer, 2013, pp. 229–250. 2, 43

[4] A. Gal, A. Mandelbaum, F. Schnitzler, A. Senderovich, M. Weidlich, Traveling Time Prediction in Scheduled Transportation with Journey Segments, Tech. rep., Technical report, Technion-Israel Institute of Technology (2014). 2

[5] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining - predicting delays in service processes, in: Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings, 2014, pp. 42–57. 2

[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. 3

[7] M. H. Ammar, S. B. Gershwin, Equivalence relations in queueing models of fork/join networks with blocking, Performance Evaluation 10 (3) (1989) 233–245. 3

[8] R. Atar, A. Mandelbaum, A. Zviran, Control of fork-join networks in heavy traffic, in: Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on, IEEE, 2012, pp. 823–830. 3, 8, 45

[9] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, S. Kadish, C. A. Bunnell, Discovery and validation of queueing networks in scheduled processes, in: Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings, 2015, pp. 417–433. 3

[10] G. Balbo, S. Bruell, M. Sereno, On the relations between bcmp queueing networks and product form solution stochastic petri nets, in: null, IEEE, 2003, p. 103. 8

[11] F. Baccelli, W. A. Massey, D. Towsley, Acyclic fork-join queuing networks, Journal of the ACM (JACM) 36 (3) (1989) 615–642. 8, 9

[12] F. P. Kelly, Networks of queues with customers of different types, Journal of Applied Probability (1975) 542–554. 9

[13] P. S. Adler, A. Mandelbaum, V. Nguyen, E. Schwerer, From project to process management: an empirically-based framework for analyzing product development time, Management Science 41 (3) (1995) 458–484. 9

[14] J. M. Harrison, Stochastic networks and activity analysis, Translations of the American Mathematical Society-Series 2 207 (2002) 53–76. 9

[15] 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. 9

[16] 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. 9, 11

[17] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011. 10, 44

[18] A. Burattin, Heuristics miner for time interval, in: Process Mining Techniques in Business Environments, Springer, 2015, pp. 85–95. 11

[19] J. F. Allen, P. J. Hayes, A common-sense theory of time., in: IJCAI, Vol. 85, 1985, pp. 528–531. 11

[20] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining - predicting delays in service processes, in: Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings, 2014, pp. 42–57. 11, 44

[21] P. Zhang, N. Serban, Discovery, visualization and performance analysis of enterprise workflow, Computational statistics & data analysis 51 (5) (2007) 2670–2687. 11

[22] 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, 44

[23] R. G. Sargent, Verification and validation of simulation models, in: S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White, M. C. Fu (Eds.), Winter Simulation Conference, WSC, 2011, pp. 183–198. 12

[24] P. J. Bickel, K. A. Doksum, Mathematical Statistics: Basic Ideas and Selected Topics, volume I, Vol. 117, CRC Press, 2015. 13

[25] T. W. Anderson, L. A. Goodman, Statistical inference about markov chains, Ann. Math. Statist. 28 (1) (1957) 89–110. 15

[26] P. Momcilovic, N. Trichakis, A. Mandelbaum, S. Kadish, C. A. Bunnell, Data-driven appointment scheduling under uncertainty, Working Paper. 15

[27] J. D. Gibbons, S. Chakraborti, Nonparametric statistical inference, Springer, 2011. 15, 18

[28] S. Maman, Uncertainty in the Demand for Service: The Case of Call Centers and Emergency Departments, Master's thesis (2009). 16

[29] J. D. Little, A proof for the queuing formula: L= λ w, Operations research 9 (3) (1961) 383–387. 19

[30] S.-H. Kim, W. Whitt, Estimating waiting times with the time-varying little's law, Probability in the Engineering and Informational Sciences 27 (04) (2013) 471–506. 19

[31] D. H. Maister, The psychology of waiting lines, Harvard Business School, 1984. 19

[32] A. Mandelbaum, W. A. Massey, M. I. Reiman, A. Stolyar, B. Rider, Queue lengths and waiting times for multiserver queues with abandonment and retrials, Telecommunication Systems 21 (2-4) (2002) 149–171. 20

[33] M. L. Pinedo, Scheduling: theory, algorithms, and systems, Springer Science & Business Media, 2012. 22, 23, 25, 26, 45

[34] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, A. Mandelbaum, S. Kadish, C. A. Bunnell, Data-driven performance analysis of scheduled processes, in: Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, 2015, pp. 35–52. 22

[35] J. R. Jackson, Scheduling a production line to minimize maximum tardiness, Tech. rep., DTIC Document (1955). 26

[36] J. K. Lenstra, A. R. Kan, P. Brucker, Complexity of machine scheduling problems, Annals of discrete mathematics 1 (1977) 343–362. 26

[37] J. Du, J. Y.-T. Leung, Minimizing total tardiness on one machine is np-hard, Mathematics of operations research 15 (3) (1990) 483–495. 27

[38] A. Farcomeni, A review of modern multiple hypothesis testing, with particular attention to the false discovery proportion, Statistical Methods in Medical Research. 41

[39] E. Arjas, T. Lehtonen, Approximating many server queues by means of single server queues, Mathematics of Operations Research 3 (3) (1978) 205–223. 42

[40] S. Halfin, W. Whitt, Heavy-traffic limits for queues with many exponential servers, Operations research 29 (3) (1981) 567–588. 42

[41] J. A. Buzacott, J. G. Shanthikumar, Stochastic Models of Manufacturing Systems, Prentice Hall, Englewood Cliffs, NJ, 1993. 43

[42] R. S. Mans, N. C. Russell, W. M. van der Aalst, A. J. Moleman, P. J. Bakker, Schedule-aware workflow management systems, in: Transactions on Petri nets and other models of concurrency IV, Springer, 2010, pp. 121–143. 44

[43] A. Rogge-Solti, W. M. P. van der Aalst, M. Weske, Discovering stochastic petri nets with arbitrary delay distributions from event logs, in: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, 2013, pp. 15–27. 44

[44] W. M. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2) (2011) 450–475. 44

[45] A. Rogge-Solti, M. Weske, Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays, in: S. Basu, C. Pautasso, L. Zhang, X. Fu (Eds.), ICSOC, Vol. 8274 of Lecture Notes in Computer Science, Springer, 2013, pp. 389–403. 44

[46] A. Rozinat, R. Mans, M. Song, W. M. P. van der Aalst, Discovering simulation models, Information Systems 34 (3) (2009) 305–327. 44

[47] A. Pika, M. T. Wynn, C. J. Fidge, A. H. ter Hofstede, M. Leyer, W. M. van der Aalst, An extensible framework for analysing resource behaviour using event logs, in: Advanced Information Systems Engineering, Springer, 2014, pp. 564–579. 44

[48] J. Nakatumba, W. M. van der Aalst, Analyzing resource behavior using process mining, in: Business Process Management Workshops, Springer, 2010, pp. 69–80. 44

[49] A. Rozinat, W. M. P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Inf. Syst. 33 (1) (2008) 64–95. 44

[50] R. P. J. C. Bose, W. M. P. van der Aalst, Process diagnostics using trace alignment: Opportunities, issues, and challenges, Inf. Syst. 37 (2) (2012) 117–141. 44

[51] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, M. Weske, Process compliance analysis based on behavioural profiles, Inf. Syst. 36 (7) (2011) 1009–1025. 44

[52] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, B. Baesens, Determining process model precision and generalization with weighted artificial negative events, IEEE Trans. Knowl. Data Eng. 26 (8) (2014) 1877–1889. 44

[53] E. R. Taghiabadi, V. Gromov, D. Fahland, W. M. P. van der Aalst, Compliance checking of data-aware and resource-aware compliance requirements, in: On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings, 2014, pp. 237–257. 44

[54] M. de Leoni, W. M. P. van der Aalst, B. F. van Dongen, Data- and resource-aware conformance checking of business processes, in: Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings, 2012, pp. 48–59. 44

[55] A. Adriansyah, B. F. van Dongen, W. M. P. van der Aalst, Conformance Checking Using Cost-Based Fitness Analysis, in: EDOC, IEEE Computer Society, 2011, pp. 55–64. 44

[56] M. Dumas, L. García-Bañuelos, Process mining reloaded: Event structures as a unified representation of process models and event logs, in: Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings, 2015, pp. 33–48. 44

[57] N. R. T. P. van Beest, M. Dumas, L. García-Bañuelos, M. L. Rosa, Log delta analysis: Interpretable differencing of business process event logs, in: Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, 2015, pp. 386–405. 44

Algorithm 2 Scheduling with Completed Cases by S' (A2)

1: $t = 0; \Upsilon^{S}(t) = \Upsilon^{S'}(t) = \emptyset$; {Initialization}
2: $\sigma =$ Order cases in an INIT order
3: Calculate $T^{S}_{\max}$ (INIT)
4: Calculate $F^{S}_{\max}$ (INIT)
5: **while** $\sigma \neq \emptyset$ **do**
6: $j = 2$; $s = 1; \Delta_{\min} = 0$.
7: **if** $\sigma(j) \in \Upsilon^{S'}(t) \cap \sigma \backslash \sigma(1)$ and $p_{\sigma(j)} \leq p_{\sigma(s)}$ **then**
8: $k = 1$
9: **else**
10: go to line 28
11: **end if**
12: **if** $k \leq j$, **then**
13: $C^{S}_{\sigma(k)} = \max\{t, r_{\sigma(j)}\} + p_{\sigma(j)} + \sum_{i=1}^{k} p_{\sigma(i)}$.
14: **else**
15: go to line 23
16: **end if**
17: **if** $C^{S}_{\sigma(k)} - d_{\sigma(k)} \leq T^{S}_{\max}$ (INIT) and $C^{S}_{\sigma(k)} - r_{\sigma(k)} \leq F^{S}_{\max}$ (INIT) **then**
18: $k = k + 1$
19: go to line 12
20: **else**
21: go to line 28
22: **end if**
23: $\Delta(j) = p_{\sigma(j)}(j-1) - \sum_{i=1}^{j-1} p_{\sigma(i)}$
24: **if** $\Delta(j) \leq \Delta_{\min}$ **then**
25: $\Delta_{\min} = \Delta(j)$
26: $s = j$
27: **end if**
28: $j = j + 1$
29: **if** $j \leq |\sigma|$ **then**
30: go to line 7
31: **end if**
32: Serve case $\sigma(s)$ with S
33: $\widetilde{t} = \max\{t, r_{\sigma(s)}\} + p_{\sigma(s)}$
34: $\Upsilon^{S}\left(\widetilde{t}\right) = \left\{\Upsilon^{S}(t) \cup \sigma(s)\right\} \setminus \left\{\Upsilon^{S'}(t) \cup \Phi\left(t, \widetilde{t}\right)\right\}$
35: $\Upsilon^{S'}\left(\widetilde{t}\right) = \left\{\Upsilon^{S'}(t) \cup \Phi\left(t, \widetilde{t}\right)\right\} \setminus \left\{\Upsilon^{S}(t) \cup \sigma(s)\right\}$
36: $\sigma = \sigma \backslash \sigma(s)$
37: $t = \widetilde{t}$
38: **end while**

Chapter 4

Discussion

We conclude with a discussion of the main results, based on the papers presented in Chapter 3. In Section 4.1 we present the findings of the first paper. The paper considers queue mining for performance analysis and prediction in single-station queueing processes. The second part (Section 4.2) extends the work on queue mining to networks of queues with predefined (scheduled) routing. In the first two articles, we compare approaches that use queueing theory results directly (model-based queue mining) and supervised queue mining (see Section 2.4.2 for the difference between the two approaches). Subsequently, in Section 4.3, we provide an overview of our findings on multi-perspective conformance checking via queueing models (third article in Chapter 3), and relate conformance checking to process improvement.

4.1 Queue Mining in Single-Station Processes

In [62], we developed queue mining techniques for single-station service processes. In particular, we introduced the problem of *online delay prediction* and provided different solutions that predict delays. We provided delay predictions for the single-class setting (homogeneous customers) as well as for multi-class setting that features different classes of customers.

We achieved the above via two types of delay predictors. The first one, predictors based on *transition systems*, which are process models that, prior to this research, had not considered the queueing perspective when used for prediction [71]. To bridge this gap, we enhanced transition systems with queueing information by extending them to *featured transition systems*. Then, we showed that featured transition systems can serve, together with an event log, as an input to regression-based machine learning techniques; the latter were used to predict delays.

As an alternative to the aforementioned approach, we presented a model-based technique for queue mining, where we use *queueing predictors* that emerge from a discovered queueing model and its underlying theories. These predictors can be roughly divided into two categories: state-dependent predictors and stationary predictors. The former are based on real-time queue lengths that a customer is facing upon arrival, while the latter are based on previously observed delays, and assume that the system does not change (much) during the visit of an arriving customer. The stationary predictors are referred to as *snapshot predictors*, since they stem from a well-established principle in queueing approximations, namely the *snapshot principle* [50].

We tested the accuracy of all predictors using real-life service logs from the telecommunication and financial sectors. Our experiments show that transition system predictors incorporating queueing information, or directly grounded in queueing models, improve accuracy by 30%–40% on average compared to plain regression-based methods. Also, we observed that the multi-class methods are superior to single-class methods in their predictive power, which is not surprising.

Finally, we reported on sensitivity analysis of the predictors with synthetic (simulation generated) data, by exploring scenarios of increasing load, various sizes, and under varying impatience. Here, methods based on transition systems, which incorporate queueing information as well as snapshot predictors, have been shown to be particularly robust across the different scenarios. Below, we provide an extended discussion of two important aspects of model-based queue mining, namely modeling granularity (single-class vs. multi-class predictors), and the effect of system load on prediction accuracy.

4.1.1 Single-Class vs. Multi-Class Predictors

The single class analysis in [62] shows a slight advantage of queueing-based methods over transition system techniques (Figure 6 in [62]). Indeed, both the difference in the predictive power and the run-time complexity of the snapshot predictors (that assume steady-state) seems appealing for answering operational questions such as 'how long will a particular customer wait?'. However, in most services nowadays, customers are divided into classes in correspondence to, e.g., sophisticated Customer Relationship Management (CRM) tools. For example, the classes can be a function of the financial value that a customer brings into the company or according to a patient's current health status.

Prediction methods should therefore accommodate 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 [59], 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 8 in [62] we learn that the performance of the VIP system is stable, whereas low priority customers experience significant changes in load and thus in delay duration. Indeed, upon moving to a dataset with multi-class characteristics, the snapshot predictors that are class independent do lack robustness. The transition system method gives rise to reasonable results, despite ignoring multiple classes. This difference emphasizes the dependence between quality of prediction and the queueing models that the predictors are based upon. On the other hand, process models without the queueing perspective do not provide insights on the customer resource interaction. However, these techniques are often robust over various datasets and scenarios. Once we have adjusted the snapshot predictor and its assumptions to the new reality where multiple types of customers exist, its performance level increased dramatically. Queue-length based approximations of delays via upper and lower bounds turn out to be useful. This implies that these predictors can be used in order to predict delays. Note that the lower bound predictor is comparable to the snapshot predictor, which supports the validity of the assumptions made in the queueing model.

4.1.2 Effects of System Load

The notion of system load (or offered load) is fundamental in service processes (see, for example, [81]). The offered load can be defined as the amount of exogenous work (in units of time) that enters the system per time unit. Alternatively, one may consider the offered load to be the resource capacity required to satisfy all incoming demand, if there were no limit on its availability. The load is reflected in queue lengths, delays, resource utilization and the probability that customers abandon. In our sensitivity analysis with synthetic logs [62], we validated queue mining methods against increasing load and stable load.

In the increasing load scenario (Scenario 1), when resource utilization increases and heavy-traffic conditions drive the system to a steady-state (of long queues and prolonged delays), the Head-Of-Line predictor performs very well. Specifically, Figure 10 in [62] shows a non-surprising result, that when a (heavy-traffic) steady state is reached, the Head-Of-Line predictor performs better than for an intermediate load. Tree-based methods perform as well

as the Head-Of-Line predictor, since delays are more predictable in steady-state; less data is therefore required for generalization. In contrast, the queueing model seems to 'miss out' the steady-state that results from heavy-traffic conditions. The worsening of the queueing model predictor is related to its insensitivity to steady-state behaviour, being a time-varying (state-dependent) predictor. For intermediate load, the queueing model predictors are comparable to Head-Of-Line and tree predictors and therefore can be considered accurate for delay prediction.

In the second scenario (Figure 11 in [62]), when service times increase but load is stable, we observe that the snapshot predictor and the transition-based method are stable in performance. On the other hand, the queueing model predictor deteriorates. We explain this phenomena by the fact that, in the second, scenario we did not scale customer patience according to the increase in service times. Scenario 2 indicates that the Head-Of-Line and the transition-system methods are invariant to an increase in service times. However, for the queueing model predictors, a refinement of experiments is required to check for their accuracy under increasing service times.

An observed phenomena in service systems is that customer patience could depend on service time. Customers are willing to wait longer for a longer service [13]. Hence, we conducted a third experiment, and in Scenario 3 scaled the patience as well. Scenario 3 is demonstrated in Figures 12 and 13 in [62]; in this scenario, the load is stable, although patience and service times increase by order of magnitude. For absolute errors, all predictors perform worse as service times increase. However, when we consider the relative error, with respect to the average delay, we observe that all predictors are stable across simulation runs. This is an expected result since. for the same load and queue-lengths, the relative prediction accuracy per predictor should remain the same. Clearly, an error of 50 seconds is substantial for an average delay of 20 seconds, but is considered low for an average delay of 900 seconds (15 minutes). To summarize, service processes operate under varying loads. Some systems never experience heavy-load and therefore queueing model predictors can be adequate. For overloaded systems, snapshot predictors and learning methods seem to perform better when predicting delays. When the load is stable, we expect an adequate performance of all predictors, regardless of the value of the load.

4.2 Queue Mining in Networks

In this part, we recap the results presented in [29]. The paper extends queue mining to networks of queues with scheduled customer routing. Schedules that drive processes are encountered in transportation (timetables), healthcare (appointment books), and manufac-

turing (material requirements planning). To demonstrate the usefulness of queue mining in networks, we present a novel approach to online prediction of bus traveling times. The approach is based on considering bus journeys as trajectories of segments, with a segment being the travel time between two consecutive stops. These segments are then treated as queueing stations, with buses being the customers. To predict traveling times, we once more combine supervised queue mining and predictors that stem from discovered queueing models. Our empirical analysis confirms that this combination indeed improves performance with respect to the two approaches, separately. This finding supports the paradigm of a hybrid approach that involves both supervised and model-based queue mining.

Below, we explain the lifting of the snapshot predictor, which serves as our prime queueing predictor, to a network of queues. Then, we go over the main empirical results, with emphasis on the hybrid approach for queue mining, which uses both machine learning and queueing analysis.

4.2.1 Snapshot Predictors in Networks

The *snapshot principle* [80, p. 187] is a heavy-traffic approximation that refers to the behavior of a queueing model under limits of its parameters, as the workload converges to capacity. We showed that snapshot-based predictors are useful when predicting delays in single-station multi-class processes [62]. In real-life settings, the heavy-traffic approximation need not be apply and thus the applicability of the snapshot principle predictors should be tested ad-hoc, when working with real-world datasets. Results of synthetic simulation runs [35] show that snapshot-based predictors are indeed appropriate for predicting delays.

In the context of buses, the snapshot principle means that a bus that passes through a segment will experience a similar traveling time as another bus that has just passed through that segment (not necessarily of the same line). Based on the above, we define a single-segment snapshot predictor, namely the Last-Bus-to-Travel-Segment (LBTS). In this setting, however, the LBTS predictor must be lifted to a network of queues. In [50], it is stated that the snapshot principle holds for networks of queues, when the particular route through this network is known in advance. Clearly, in scheduled transportation such as buses, this is the case as the order of stops (and segments) is predefined. Therefore, we define a multi-segment (network) snapshot predictor that we refer to as the Last-Bus-to-Travel-Network (LBTN), to have an additive form. In other words, if a bus is to travel through a sequence of n stops, $\langle \omega_1, ..., \omega_n \rangle$, the LBTN is the sum of the LBTS predictors for each $\omega_i, i = 1, \ldots, n$. We hypothesized that the snapshot predictor performs better whenever recent buses are in time proximity to the current journey.

4.2.2 Main Results

Below, we go over our main results of the experimental evaluation [29]. The experiments show that prediction methods that combine the snapshot predictor and regression tree techniques are superior, in terms of prediction quality, over performing separately either snapshot predictors or regression tree methods. Specifically, Table 3 in [29] summarizes the prediction error in terms of root-mean squared error, mean absolute relative error, and median absolute relative error.

The most accurate techniques result from the combination of the snapshot predictor and methods of gradient boosting (based on regression trees). This combination is in the spirit of *model adaptation* methods for supervised queue mining, as described in Section 2.4.2. Further, we found that the prediction error increases with the number of bus stops per journey (Figure 6 in [29]). However, the relative error is stable over trip lengths, which indicates that prediction accuracy does not deteriorate with respect to the length of journeys. This finding implies that the path lengths for a traveling customer does not have a negative influence on the relative prediction error.

Somewhat surprisingly, the snapshot predictor performance does not deteriorate for longer trips, therefore contradicting the hypothesis that the snapshot predictor would be more precise for journeys with higher temporal proximity to the current journey. This shows that the snapshot predictor is rather robust, and that unless drastic changes in the load occur, it is a stable predictor. Lastlys, Figure 9 in [29] shows that prediction accuracy is negatively correlated with the number of buses traveling through the city (a proxy for system load). This finding is reasonable, since in peak hours the variance in traveling durations increases, which makes the prediction task more difficult for a given fixed model.

4.3 Conformance Checking with Queue Mining

In [64], we provided methods for conformance checking and performance improvement of *scheduled multi-stage service processes*. To assess the conformance of a schedule of a process to its actual execution, we presented an approach based on process discovery and statistical comparison of two Fork/Join networks. Specifically, we discover a deterministic queueing network from schedule, and a stochastic network from the actual executions. Then, we provide a comparison technique to evaluate the differences between the two networks. We assume that capturing deviations in building blocks of the two networks, would lead to capturing deviations between schedule and actual executions.

Further, the discovered Fork/Join network was then used to improve the underlying

scheduled process via techniques of mathematical scheduling. Specifically, we developed theoretical results that guarantee non-decreasing performance measures, such as tardiness and flow time, when ordering cases for concurrent processing. We evaluated the approach in two steps using real-world data from an outpatient hospital. First, we showcased how our approach for conformance checking leads to the identification of operational bottlenecks and that it supports the analysis of the root-causes of these bottlenecks. Second, we demonstrated the value of our process improvement approach by simulating alternative scheduling realities that used case orderings as recommended by our algorithms. The experiments resulted in a 20%–40% improvement for the various measures, with respect to a baseline of the actual ordering of cases.

In the remainder of this section, we discuss the main results presented in [64]. Moreover, we address the connection between conformance checking and process improvement.

4.3.1 Root-Cause Analysis

Our experiments in [64] were based on data from the Dana-Farber Cancer Institute. Specifically, we focused on the patient treatment process, as described in Figure 2.1. We started by applying techniques for checking operational conformance, realizing that actual executions deviate from schedule. This is observed in Figure 5 in [64], as the distribution of the delay between vital signs and infusion exhibits a non-negligible tail, with an average value of 25 minutes. According to schedule, there should be no delay between these two stations. It becomes apparent that the reason for the delay is rooted in the pharmacy station (Figure 2.1), as it does not supply the chemotherapy medications on time (with respect to patient schedules).

A comparison of the two Fork/Join networks showed that the production times of chemotherapeutic drugs are not deviating from the scheduled production time, which is 30 minutes (Figure 6 in [64]), and has a relatively small variance. However, when we compared the drug production policy in the pharmacy, we found out that it is insensitive to the current state, or to the due dates dictated by the schedule (Figure 7 in [64]). This leads to the conclusion that the deviation in production policy of the pharmacy is the root-cause for the observed delay in Figure 5. The above case study demonstrates the usefulness of our conformance checking approach for detecting root-causes of deviations between what was scheduled and actual process executions.

4.3.2 Process Improvement

As we mentioned earlier, the pharmacy unit was causing deviations from schedule. Having found the non-conforming part, we propose a solution that would increase conformance and improve patient flow. Given that the pharmacy is an independent unit that cannot be controlled, we hypothesize that the case sequencing in the vital signs station can be improved, without reducing the tardiness of drug provision, while lowering the total complete times for all patients. The improvement idea is as follows. If a patient is next to enter infusion according to schedule, and her drug is not ready yet, a different patient (whose drug is ready) can be preempted to reduce the total completion time. We have formally demonstrated two scheduling algorithms [64] that guarantee to outperform the earliest-due-date first (EDD) policy, which is assumed in the vital signs station. The first algorithm uses a mixture of the EDD and first-come first-served (FCFS) algorithms to outperform EDD. The second algorithm employs the state of the F/J network, i.e. the number of waiting customers in the synchronization queues, to better order the cases. This leads to an improvement in sum of completion times, while guaranteeing that all deadlines are met.

To test the two proposed scheduling algorithms we ran a data-based simulation experiment. Specifically, we simulated the process described in Figure 1.1, by using real data values for vital sign durations, pharmacy production times and policy. The only alternation that we introduce (w.r.t. the data) is changing the patient sequencing policy in the vital signs station. Not surprisingly, we observe that, with respect to the three performance measures (tardiness, complete time and flow time), both algorithms improve over EDD (schedule-driven policy). Furthermore, we observe a large improvement for median flow time and median tardiness, as well as an improvement in all measures by considering the synchronization between two queueing stations.

Figure 8 presents improvement over the baseline, in percentage, with negative values corresponding to improvement in performance. For example, the value of -20.66% for F_{median} corresponds to improvement of 20.66% with respect to the simulated median flow when actual sequencing is applied.

Moving from all patients to specific diagnoses, Figure 9 shows a similar analysis for hematology malignancy patients. These patients are treated in a separate disease center, and have dedicated clinical assistants. We observe that the behavior of the results is similar to the overall population, yet with larger improvement. We believe that this improvement stems from a higher conceptual conformance of the single-server assumption for these patients. Finally, Figure 10 presents the improvement in a sum of completion times as a day-of-week function. Wednesdays are known to be the most loaded days in the Dana-Farber Cancer

Institute (Figure 11). We observe that the highest improvement rate is obtained for the more loaded days. This result puts forward the importance of correct sequencing, especially on heavily loaded days.

To conclude, there is a direct relation between the conformance of the actual executions and the schedule to process improvement. The non-conforming parts often indicate improvable process management, thus serving as a detection mechanism for parts of the process that can be improved. In this work, we demonstrated this connection via a solution that combines data-driven analysis and scheduling.

Other operations management disciplines and solutions can be considered for solving various possible deviations from schedule. For example, service time reduction can be treated via process re-engineering [16], while improving arrival rate prediction and update can be handled via statistical techniques for arrival prediction [13].

Bibliography

[1] Paul S Adler, Avishai Mandelbaum, Viên Nguyen, and Elizabeth Schwerer. From project to process management: an empirically-based framework for analyzing product development time. *Management Science*, 41(3):458–484, 1995. 12

[2] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *EDOC*, pages 55–64. IEEE Computer Society, 2011. ISBN 978-1-4577-0362-1. 4

[3] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment Based Precision Checking. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 137–149. Springer, 2012. ISBN 978-3-642-36284-2. 4

[4] James F Allen and Patrick J Hayes. A common-sense theory of time. In *IJCAI*, volume 85, pages 528–531, 1985. 17

[5] Mor Armony, Shlomo Israelit, Avishai Mandelbaum, Yariv N Marmor, Yulia Tseytlin, and Galit B Yom-Tov. On patient flow in hospitals: A data-based queueing-science perspective. *Stochastic Systems*, 5(1):146–194, 2015. 9

[6] Rami Atar, Avishai Mandelbaum, and Asaf Zviran. Control of Fork-Join Networks in heavy traffic. In *Allerton Conference*, pages 823–830. IEEE, 2012. ISBN 978-1-4673-4537-8. 13

[7] David Azriel, Paul D Feigin, and Avishai Mandelbaum. Erlang S: A data-based model of servers in queueing networks. Technical report, Working paper, 2014. 9

[8] François Baccelli, William A Massey, and Don Towsley. Acyclic fork-join queuing networks. *Journal of the ACM (JACM)*, 36(3):615–642, 1989. 11, 12, 13

[9] Falko Bause and Pieter S Kritzinger. *Stochastic Petri Nets*. Springer, 2002. 2

[10] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications; 2nd Edition*. Wiley, 2006. ISBN 978-0-471-56525-3. 13, 18

[11] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.*, 37(2):117–141, 2012. 5

[12] Maury Bramson. *Stability of Queueing Networks*. Springer, 2008. 19

[13] Lawrence Brown, Noah Gans, Avishai Mandelbaum, Anat Sakov, Haipeng Shen, Sergey Zeltyn, and Linda Zhao. Statistical analysis of a telephone call center. *Journal of the American Statistical Association*, 100(469):36–50, 2005. doi: 10.1198/016214504000001808. URL http://www.tandfonline.com/doi/abs/10.1198/016214504000001808. 12, 18, 155, 160

[14] Joos CAM Buijs, Boudewijn F van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012. 3

[15] Andrea Burattin. Heuristics miner for time interval. In *Process Mining Techniques in Business Environments*, pages 85–95. Springer, 2015. 17

[16] John A Buzacott. Commonalities in reengineered business processes: Models and issues. *Management Science*, 42(5):768–782, 1996. 160

[17] John A Buzacott and J George Shanthikumar. *Stochastic Models of Manufacturing Systems*, volume 4. Prentice Hall Englewood Cliffs, NJ, 1993. 9

[18] Hong Chen and David D Yao. *Fundamentals of Queueing Networks: Performance, Asymptotics, and Optimization*, volume 46. Springer Science & Business Media, 2013. 18

[19] Massimiliano De Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management*, pages 113–129. Springer, 2013. 8

[20] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1454–1461. ACM, 2013. 7, 8

[21] Massimiliano De Leoni, Wil M. P. van der Aalst, and Boudewijn F van Dongen. Data- and resource-aware conformance checking of business processes. In *Business Information Systems*, pages 48–59. Springer, 2012. 5, 8

[22] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235–257, 2016. 6, 8

[23] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007. 3

[24] Marlon Dumas and Luciano García-Bañuelos. Process mining reloaded: Event structures as a unified representation of process models and event logs. In *Application and Theory of Petri Nets and Concurrency*, pages 33–48. Springer, 2015. 4

[25] Dirk Fahland and Wil M. P. Van Der Aalst. Simplifying discovered process models in a controlled manner. *Information Systems*, 38(4):585–605, 2013. 5

[26] Dirk Fahland and Wil M .P. van der Aalst. Model repair–aligning process models to reality. *Information Systems*, 47:220–243, 2015. 5

[27] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering context-aware models for predicting business process performances. In *On the Move to Meaningful Internet Systems: OTM 2012*, pages 287–304. Springer, 2012. 8

[28] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001. doi: 10.1214/aos/1013203451. URL http://dx.doi.org/10.1214/aos/1013203451. 20

[29] Avigdor Gal, Avishai Mandelbaum, François Schnitzler, Arik Senderovich, and Matthias Weidlich. Traveling time prediction in scheduled transportation with journey segments. *Information Systems*, 2015. 6, 7, 8, 9, 18, 19, 20, 21, 155, 157

[30] Peter W Glynn and Donald L Iglehart. Simulation methods for queues: An overview. *Queueing systems*, 3(3):221–255, 1988. 18

[31] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007. ISBN 978-3-540-75182-3. 3

[32] Peter J Haas. *Stochastic Petri Nets: Modelling, Stability, Simulation*. Springer, 2002. 6

[33] J Michael Harrison. Stochastic networks and activity analysis. *Translations of the American Mathematical Society-Series 2*, 207:53–76, 2002. 12

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 7, 16

[35] Rouba Ibrahim and Ward Whitt. Real-time delay estimation based on delay history. *Manufacturing and Service Operations Management*, 11(3):397–415, 2009. doi: 10.1287/msom.1080.0223. URL http://msom.journal.informs.org/content/11/3/397.abstract. 18, 156

[36] James R Jackson. Networks of waiting lines. *Operations research*, 5(4):518–521, 1957. 19

[37] Frank P Kelly. Networks of queues with customers of different types. *Journal of Applied Probability*, 12(3):542–554, 1975. 12, 19

[38] David 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):338–354, 1953. URL http://www.jstor.org/stable/2236285. 12

[39] Sander JJ Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In *Application and Theory of Petri Nets and Concurrency*, pages 311–329. Springer, 2013. 3

[40] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 297–313, 2015. doi: 10.1007/978-3-319-23063-4_21. URL http://dx.doi.org/10.1007/978-3-319-23063-4_21. 6, 8

[41] Avishai Mandelbaum. Service engineering (science, management): A subjective view. Technical report, Technical report, Technion-Israel Institute of Technology, 2007. 9

[42] Avishai Mandelbaum and Sergey Zeltyn. Data-stories about (im)patient customers in tele-queues. *Queueing Systems*, 75(2-4):115–146, 2013. 9

[43] Felix Mannhardt, Massimiliano de Leoni, Hajo A Reijers, and Wil M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016. 5, 7, 8

[44] Laura Măruşter and Nick RTP van Beest. Redesigning business processes: a methodology based on simulation and process mining techniques. *Knowledge and Information Systems*, 21(3):267–297, 2009. 6, 8

[45] IEEE Task Force Onf Process Mining. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011. ISBN 978-3-642-28107-5. 5

[46] Joyce Nakatumba and Wil M. P. van der Aalst. Analyzing resource behavior using process mining. In Stefanie Rinderle-Ma, Shazia Wasim Sadiq, and Frank Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 69–80. Springer, 2009. ISBN 978-3-642-12185-2. 5

[47] Business Process Model OMG. Notation (bpmn) version 2.0 (2011). *Available on: http://www. omg. org/spec/BPMN/2.0*, 2011. 2

[48] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. *CoRR*, abs/1602.07566, 2016. URL http://arxiv.org/abs/1602.07566. 6, 8

[49] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. Where Did I Misbehave? Diagnostic Information in Compliance Checking. In Alistair P. Barros, Avigdor Gal, and Ekkart Kindler, editors, *BPM*, volume 7481 of *Lecture Notes in Computer Science*, pages 262–278. Springer, 2012. ISBN 978-3-642-32884-8. 4

[50] Martin I Reiman and Burton Simon. A network of priority queues in heavy traffic: One bottleneck station. *Queueing Systems*, 6(1):33–57, 1990. 153, 156

[51] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *ICSOC*, volume 8274 of *Lecture Notes in Computer Science*, pages 389–403. Springer, 2013. ISBN 978-3-642-45004-4. 5

[52] Andreas Rogge-Solti and Mathias Weske. Prediction of business process durations using non-markovian stochastic petri nets. *Inf. Syst.*, 54:1–14, 2015. doi: 10.1016/j.is.2015.04.004. URL http://dx.doi.org/10.1016/j.is.2015.04.004. 6, 8

[53] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in prom. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer, 2006. ISBN 3-540-38901-6. 5

[54] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008. 4

[55] Anne Rozinat, RS Mans, Minseok Song, and Wil M. P. van der Aalst. Discovering simulation models. *Information Systems*, 34(3):305–327, 2009. 5, 6, 8

[56] Reuven Y. Rubinstein. *Monte Carlo Optimization, Simulation, and Sensitivity of Queueing Networks*. John Wiley & Sons, Inc., 1986. 18

[57] Arik Senderovich. Service analysis and simulation in process mining. In *Business Process Management Workshops*, pages 578–581. Springer, 2014. 5

[58] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Mining resource scheduling protocols. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2014. ISBN 978-3-319-10171-2. doi: 10.1007/978-3-319-10172-9. URL http://dx.doi.org/10.1007/978-3-319-10172-9. 18

[59] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Mining resource scheduling protocols. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2014. ISBN 978-3-319-10171-2. doi:

10.1007/978-3-319-10172-9. URL http://dx.doi.org/10.1007/978-3-319-10172-9. 154

[60] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Mining resource scheduling protocols. In *Business Process Management*, pages 200–216. Springer, 2014. 8, 9, 19

[61] Arik Senderovich, Andreas Rogge-Solti, Avigdor Gal, Jan Mendling, Avishai Mandelbaum, Sarah Kadish, and Craig A Bunnell. Data-driven performance analysis of scheduled processes. In *Business Process Management*, pages 35–52. Springer, 2015. 6, 8, 9

[62] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Queue mining for delay prediction in multi-class service processes. *Information Systems*, 53: 278–295, 2015. 7, 8, 9, 10, 18, 19, 20, 152, 153, 154, 155, 156

[63] Arik Senderovich, Matthias Weidlich, Avigdor Gal, Avishai Mandelbaum, Sarah Kadish, and Craig A Bunnell. Discovery and validation of queueing networks in scheduled processes. In *Advanced Information Systems Engineering*, pages 417–433. Springer, 2015. 9, 19

[64] Arik Senderovich, Matthias Weidlich, Liron Yedidsion, Avigdor Gal, Avishai Mandelbaum, Sarah Kadish, and Craig A Bunnell. Conformance checking and performance improvement in scheduled processes: A queueing-network perspective. *Information Systems*, 2016. 5, 7, 8, 9, 11, 16, 17, 18, 157, 158, 159

[65] Wil M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, pages 407–426. Springer, 1997. 2

[66] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. ISBN 978-3-642-19344-6. 1, 2, 3, 4, 5, 14, 17

[67] Wil M. P. van der Aalst. Process mining: overview and opportunities. *ACM Trans. Management Inf. Syst.*, 3(2):7, 2012. 1, 5

[68] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9): 1128–1142, 2004. 2

[69] Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In

Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *OTM Conferences (1)*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2005. ISBN 3-540-29736-7. 4

[70] Wil M. P. van der Aalst, Hajo A. Reijers, and Minseok Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593, 2005. 5

[71] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011. 1, 5, 6, 8, 152

[72] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. 8

[73] Boudewijn F van Dongen, RA Crooy, and Wil M. P. van der Aalst. Cycle time prediction: When will this case finally be finished? In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 319–336. Springer, 2008. 5

[74] Maikel L van Eck, Xixi Lu, Sander JJ Leemans, and Wil M. P. van der Aalst. Pmˆ2: A process mining project methodology. In *Advanced Information Systems Engineering*, pages 297–313. Springer, 2015. 5

[75] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. A comprehensive benchmarking framework (cobefra) for conformance analysis between procedural process models and event logs in prom. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, 16-19 April, 2013*, pages 254–261, 2013. doi: 10.1109/CIDM.2013.6597244. URL http://dx.doi.org/10.1109/CIDM.2013.6597244. 5

[76] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible Heuristics Miner (FHM). In *CIDM*, pages 310–317. IEEE, 2011. ISBN 978-1-4244-9925-0. 3

[77] Anton JMM Weijters and Wil M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10 (2):151–162, 2003. 3

[78] W. Whitt. Predicting queueing delays. *Management Science*, 45(6):870–888, 1999. 19

[79] Ward Whitt. The queueing network analyzer. *Bell System Technical Journal*, 62(9): 2779–2815, 1983. 18

[80] Ward Whitt. *Stochastic-Process Limits: An Introduction to Stochastic-Process Limits and their Application to Queues*. Springer, 2002. 156

[81] Ward Whitt. Om forumoffered load analysis for staffing. *Manufacturing & Service Operations Management*, 15(2):166–169, 2013. 154

[82] Ping Zhang and Nicoleta Serban. Discovery, visualization and performance analysis of enterprise workflow. *Computational statistics & data analysis*, 51(5):2670–2687, 2007. 18

כריית תורים:
היבטי שירות בכריית תהליכים

אריק סנדרוביץ'

כריית תורים:
היבטי שירות בכריית תהליכים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה בהנדסת ניהול מידע

אריק סנדרוביץ'

הוגש לסנט הטכניון - מכון טכנולוגי לישראל
אלול התשע"ו, חיפה, ספטמבר 2016

המחקר נעשה בהנחיה משותפת של פרופ׳ אביגדור גל
ופרופ׳ אבישי מנדלבאום בפקולטה להנדסת תעשיה
וניהול.

אני מודה לטכניון על התמיכה הנדיבה בהשתלמותי.

<u>תקציר</u>

תהליכים עסקיים בעולם המודרני נתמכים על ידי מערכות מידע המסייעות לתכנון, מידול, ביצוע וניטור תהליכים בזמן אמת. בנוסף, מערכות מידע אלו מתעדות את כלל הפעולות והתנועות הקשורות לזמן הריצה של תהליכים עסקיים ושומרות את הנתונים במאגרי נתונים הנקראים *יומני רישום (Event Logs)*. *כריית תהליכים (Process Mining)* הינו תחום מחקר אשר שואף לגלות מידע שימושי על תהליכים עסקיים מיומני רישום אלו תוך שימוש בשיטות מתקדמות מתחומי כריית נתונים וניתוח תהליכים. בפרט, ניתן לדמות כריית תהליכים לגשר בין תחומי מחקר המתמקדים בניתוח נתונים (כגון: למידה חישובית וכריית נתונים) ותחומים העוסקים בשיפור תהליכים (כגון: חקר ביצועים וניהול תהליכים עסקיים).

בעבודת מחקר זו אנו מתמקדים בפיתוח שיטות וכלים לכריית תהליכים הנותנות מענה לשאלות תפעוליות, והעוסקות בניתוח פרספקטיבות הזמן והמשאב בתהליך העסקי. דוגמאות לשאלות תפעוליות אלו הינן: "האם התהליך מתבצע כמתוכנן בהיבטי זמן ומשאבים?", "כמה זמן ייקח ללקוח שהחל בביצוע התהליך לסיימו?" ו-"כיצד שינוי רמות איוש של נותני השירות ישפיע על רמות השירות הקשורות לתהליך?".

בעבודות שקדמו לעבודת מחקר זו, תשובות לשאלות לעיל ניתנו ללא התחשבות בתלויות בין המקרים השונים (או הלקוחות) אשר נמצאים במערכת בו-זמנית. למשל, שיטות עדכניות לחיזוי זמני שהיה של לקוחות במערכת השתמשו רק במידע היסטורי של הלקוח עליו שואלים את שאלת החיזוי, ללא התחשבות בכלל הלקוחות השוהים במערכת. הנחת אי-תלות זו בין הלקוחות השונים הינה מתקבלת על הדעת בתהליכים בהם אין "תחרות" על משאבים משותפים בין הלקוחות השונים. עם זאת, בתהליכי שירות, "תחרות" זו, הגורמת להיווצרות תורי משאבים, הינה עניין שבשגרה. כך לקוחות בחברות סלולר מחכים על הקו עד מענה של נציג שירות, חולים המגיעים לחדר מיון מחכים לרופאים ואחיות עד שאלה יתפנו מטיפול בלקוחות אחרים ואוטובוסים הנוסעים בעיר מחכים בתור לכלי רכב אחרים על מנת לעבור בצמתים ומקטעי כביש. הטענה העיקרית של עבודת המחקר הנוכחית היא כי על פתרונות של כריית תהליכים להתחשב בתלות בין הלקוחות השונים, כלומר בתורים שנוצרים בהמתנה למשאבים משותפים.

התרומה העיקרית של עבודת המחקר הינו פיתוח של שיטות לכריית תורים אשר משלבות מידול וניתוח מתורת התורים (תחום מענף חקר ביצועים), שיטות מסטטיסטיקה ואלגוריתמים לכריית נתונים על מנת לתת מענה לשאלות התפעוליות שהזכרנו לעיל.

לשם הדגמת הערך של *שיטות לכריית תורים (Queue Mining)* ביצענו ניסויים המבוססים על שלושה מסדי נתונים של מערכות שירות: נתוני מוקד שירות טלפוני מבנק ישראלי, נתוני מערכת תחבורה מחברת האוטובוסים של העיר דאבלין באירלנד ונתונים מבית חולים לטיפולי סרטן בבוסטון שבארצות הברית. אנו מראים כי שיטות

לכריית תורים מספקות פתרונות מדויקים יותר משיטות קיימות בכריית תהליכים לבעיות התפעוליות, משמשות כלי לאיתור צווארי בקבוק בתהליכים ואף מאפשרות מגוון שיפורים בניהול התהליך העסקי.

התיזה הינה אסופה של שלושה מאמרי עיתון. להלן פירוט קצר של התוכן והממצאים של שלושת המאמרים. במאמר הראשון, *Queue mining: Queue Mining for delay prediction in multi-class service processes*, אנו מציגים את מושג כריית התורים ושתי שיטות למידול התורים מנתונים. שיטה אחת משתמשת בלמידת מכונה (*Machine Learning*) ובייצוג התהליך בצורה של מערכת מצבים (*Transition System*). למעשה, השיטה מציגה הרחבה של שיטות קיימות בכריית תהליכים, על ידי הרחבה למאפייני תורים. השיטה השנייה לכריית תורים הינה שיטה מבוססת מודל תורים. במאמר אנו ממדלים את התהליך בעזרת מודל תורים של תחנה יחידה עם סוגי לקוחות מרובים. אנו מראים באופן אמפירי כי שתי השיטות תורמות ליכולת ניתוח התהליך מעבר למה שקיים בספרות. הניסויים שלנו מבוססים על נתונים המגיעים ממוקד שירות טלפוני של בנק ישראלי גדול. נתונים אלו מתאימים למודל התחנה היחידה שאנו מניחים. במאמר אנו מדגישים את ההבדל בין שתי הגישות לכריית תורים (למידת מכונה מול מבוססת מודל).

במאמר השני, *Traveling Time Prediction in Scheduled Transportation with Journey Segments*, אנו מרחיבים את שיטות כריית התורים לרשת תורים בעלת ניתוב לקוחות ידוע מראש (*Scheduled Network*). אנו מראים שתי שיטות חדשות לכריית תורים ברשתות אלו: האחת מבוססת קידוד מאפיינים (*Features*) למודלים של עצי רגרסיה, מודלים מוכרים בתחום למידת מכונה. השיטה השנייה הינה מבוססת מודל ומניחה מודל תורים מסוג רשת תורים (*Queueing Network*). אנו מדגימים במאמר כי השיטות המוצעות משפרות את החיזוי מעבר לשיטות קיימות בלמידת מכונה ובתורת התורים. הניסויים מבוססים על נתוני אוטובוסים (האוטובוסים הינם הלקוחות ברשת, ומקטעי הכביש הינם השרתים) מהעיר דאבלין אשר באירלנד.

במאמר השלישי, *Conformance checking and performance improvement in scheduled processes: A queueing-network perspective*, אנו מציגים שיטה להשוואה בין התהליך המתוכנן והתהליך בפועל. כדוגמה מייצגת, אנו משתמשים בתהליך טיפול יום בחולי סרטן בבית החולים Dana-Farber Cancer Institute (DFCI) שבבוסטון, ארצות הברית. השיטה להשוואה משתמשת בייצוג התהליך המתוכנן כרשת תורים אחת, ובתהליך שבוצע בפועל כפי שהוקלט בנתונים כרשת תורים שניה. שתי הרשתות מגיעות מנתונים: הראשונה מה-Schedule והשניה מנתונים המוקלטים באופן אוטומטי בעת ביצוע התהליך. לאחר זיהוי אוטומטי של ההבדלים בין התהליך המתוכנן לבין ביצוע, המתבצעת על ידי השוואות סטטיסטיות בין שתי רשתות התורים, אנו מראים שיטה לשיפור התהליך במקומות בהן התגלו בעיות תפעוליות. שיפור התהליך אותו אנו מראים במאמר מבוסס על שיטות של תזמון (Scheduling). אנו מראים את מועילות השיטות שפותחו במאמר בהתבסס על נתוני אמת שמגיעים מבית החולים DFCI.