Information Systems 62 (2016) 185–206

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Conformance checking and performance improvement in scheduled processes: A queueing-network perspective

Arik Senderovich [a,*], Matthias Weidlich [b,*], Liron Yedidsion [a], Avigdor Gal [a], Avishai Mandelbaum [a], Sarah Kadish [c], Craig A. Bunnell [c]

[a] Technion - Israel Institute of Technology, Haifa, Israel
[b] Humboldt-Universität zu Berlin, Berlin, Germany
[c] Dana-Farber Cancer Institute, Boston, MA, United States

ARTICLE INFO

Available online 28 January 2016

Keywords:
Scheduled processes
Conformance checking
Process improvement
Queueing networks
Process mining
Scheduling
Statistical inference

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.

* Corresponding authors.
E-mail addresses: sariks@tx.technion.ac.il (A. Senderovich), matthias.weidlich@hu-berlin.de (M. Weidlich), lirony@ie.technion.ac.il (L. Yedidsion), avigal@ie.technion.ac.il (A. Gal), avim@ie.technion.ac.il (A. Mandelbaum), sarah_kadish@dfci.harvard.edu (S. Kadish), craig_bunnell@dfci.harvard.edu (C.A. Bunnell).

http://dx.doi.org/10.1016/j.is.2016.01.002
0306-4379/

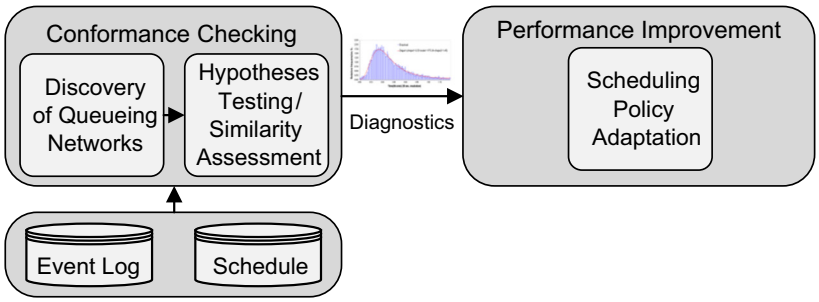


Fig. 1. An outline of our approach.

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 cancelations 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 Fig. 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 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:

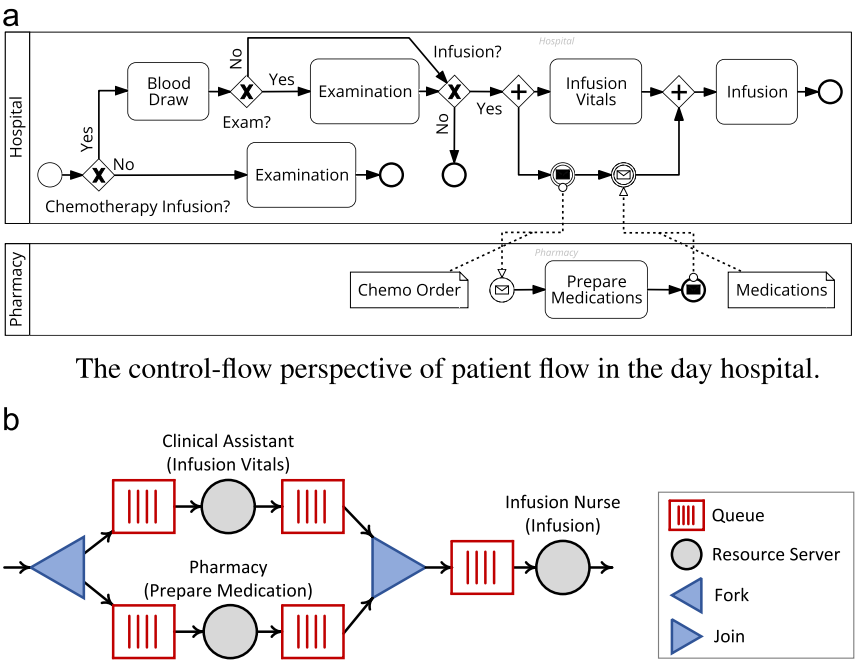
(1) We present a method to assess the conformance of a schedule and the actual process execution based on queueing networks. By means of statistical inference and similarity assessment, we identify operational deviations along with their root-causes in terms of violated assumptions underlying the scheduling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

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

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



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

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

Fig. 2. Patient flow in Dana-Farber Cancer Institute.

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. Fig. 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 canceled, after examination or after measuring infusion vital signs.

For a specific scheduled part of the aforementioned chemotherapy infusion process, Fig. 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 analyze 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 predefined 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 canceled, 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 composed 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 min. 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).
- $\delta_p: T \to \mathcal{D} \subseteq \mathbb{N}^+$ assigns a duration to a task (e.g. predefined 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 canceled 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.

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

Case id	Activity	Resource type	Start time	Duration (min)
111	Blood draw	Phlebotomist	7:30:00	15
111	Exam	Physician	9:30:00	30
111	Chemo. infusion	Inf. nurse	10:30:00	180
222	Blood draw	Nurse	9:30:00	15
222	Chemo. infusion	Inf. nurse	11:00:00	120

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

Case id	Activity	Resource type	Start time	Duration (min)
111	Blood draw	Phlebotomist	7:12:00	8
111	Exam	Physician	9:30:00	42
111	Vitals	C. Assistant	10:12:00	4
111	Chemo. product.	Pharmacy	10:12:00	35
111	Chemo. infusion	Inf. nurse	10:47:00	167
222	Blood draw	Nurse	9:26:00	25

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 synchronization 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, behavioral 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 behavior 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 min or 30 min and, depending on the duration, the scheduling is implemented differently. As such, in this example, there are three customer classes (consultation, examination in 15 min, examination in 30 min).

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 Fig. 2b. 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 in Fig. 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 characterization 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, Chapter 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.

Characterizing 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 a 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

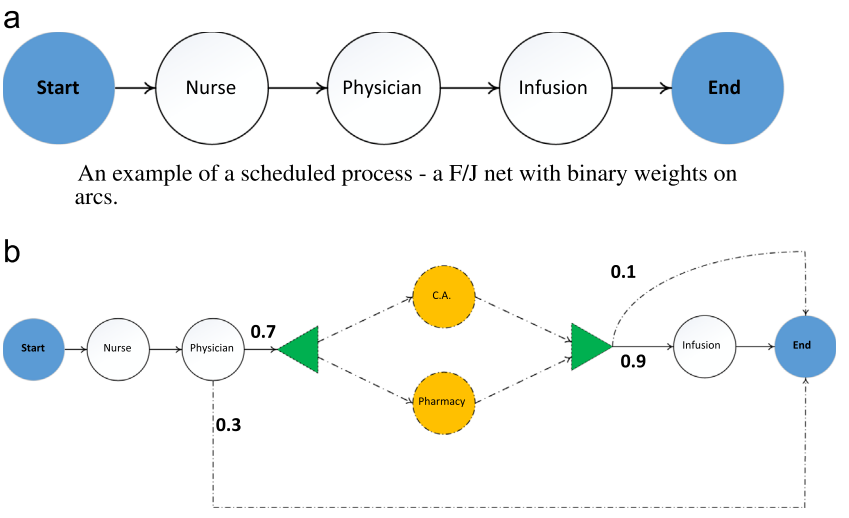


Fig. 3. Scheduled and actual F/J network structures.

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 and 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 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, Fig. 3a and b presents 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:

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

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 $T_I = \{t \in T_A | \rho_a(t) \in S_I\}$ 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:

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

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):

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

Based on the sampled tardiness values D_e, we realize 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 25th and the 75th).

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 nonhomogeneous 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, $k \in H = \{k_1, \ldots, k_m\}$, 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 (similar to the stationary case):

$$H_{0,k_i}: M_d^{k_i} = 0$$

$$H_{1,k_i}: \text{otherwise}. \tag{6}$$

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

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

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

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

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 analyze no-shows scheduled for a server $s \in S_l$ 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:

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

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 $R_l \subseteq R$, for which the server nodes are in S_l, 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 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 'centered' 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:

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

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:

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

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

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

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 set of tasks $\{t_1, \ldots, t_n\}$ waiting in the respective resource queue at time k:

$$P_s(\{t_1, \ldots, t_n\}, k) = \underset{t' \in \{t_1,\ldots,t_n\}, \tau_p(t') < k}{\arg\min} t'. \quad (13)$$

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 Eq. (13). Then, we define a similarity measure that quantifies the level of compliance to policy P_s:

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

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:

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

with $V(t) = \{t' \in T_P | \xi(t') = t \wedge t' \prec_P t\}$ 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].

Synchronization 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 Fig. 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 synchronization delays for a task that has been scheduled ($t \in T_P$) by $D_X(t)$:

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

Note that any reduction in synchronization delays with respect to schedule is considered to be positive. Hence, the above measure considers only the deviations, where the synchronization 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 synchronization 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 differences for all tasks is 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 Fig. 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:

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

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 (EDD) first 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 Fig. 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.

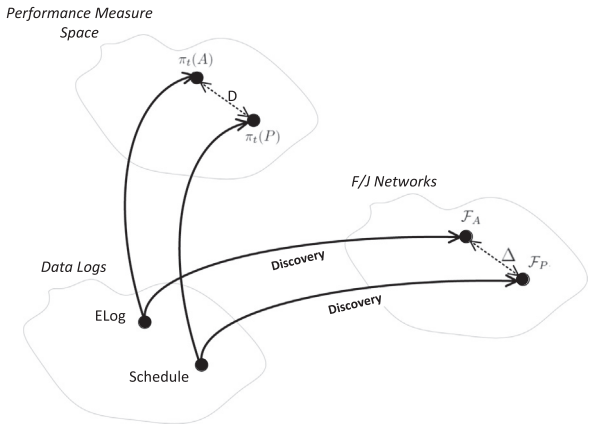


Fig. 4. The assumption of continuous conformance.

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 Fig. 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 $S_1, \ldots, S_M$ 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 $\{S_0, S_1, \ldots, S_M\}$.

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 simplicity 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 $s \in \{S_0, S_1, \ldots, S_M\}$. The completion times of a case ξ of a resource server s are 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:

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

It is worth noting that the sequence and processing times on machines $S_1, \ldots, S_M$ 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 spent while 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: $T_{\max} = \max_{\xi \in \Xi}\{T_\xi\}$, maximal flow time: $F_{\max} = \max_{\xi \in \Xi}\{F_\xi\}$, 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 $S_1, \ldots, S_M$ are a priori unknown, Eq. (19) could be rewritten as follows:

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

where S' is an alternative single machine with $C_\xi^{S'} = \max_{i \in \{1,\ldots,M\}}\left\{C_\xi^{S_i}\right\}$. 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,

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

$$F^{S}_{\max} = \max_{\xi \in \Xi}\left\{F^{S}_{\xi}\right\} = \max_{\xi \in \Xi}\left\{C^{S,C}_{\xi} - r^{C}_{\xi}\right\} \quad (24)$$

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 [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

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

Thus,

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

$\sum_{\xi \in \Xi} C^{S'}_{\xi}$ is a constant and minimizing $\sum_{\xi \in \Xi} \max\left\{C^{S}_{\xi} - C^{S'}_{\xi}, 0\right\}$ is equivalent to minimizing $\sum_{\xi \in \Xi} T_{\xi}$ with a single server where $\forall \xi \in \Xi: C^{S'}_{\xi} = d_{\xi}$. 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, starts 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.

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 $T^{S}_{\xi}(\text{A1}')$ 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** $\Gamma \neq \varnothing$ **do**
7: $\Psi = \{j: j < i, d_{(j)} \geq d_{(i)}\}$
8: $k = \arg\max_{j \in \Psi}\{d_{(j)}\}$
9: Transfer case (k) immediately following case (i)
10: Calculate $T^{S}_{\xi}(\text{A1}')$ for all cases.
11: $\Gamma = \left\{\xi: T^{S}_{[\xi]}(\text{A1}') > T^{S}_{\max}(\text{EDD})\right\}$
12: $i = \min_{\xi \in \Gamma}\{j\}$
13: **end while**

Line 1 sequences the cases in an EDD order of release time and records $T^{S}_{\max}(\text{EDD})$. 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:

$$T^{A}_{\max}(\text{A1}') \leq T^{A}_{\max}(\text{EDD}). \quad (25)$$

Γ is the set of positions of all cases whose tardiness exceeds $T^{S}_{\max}(\text{EDD})$. 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.

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 case, a, which exceeds $T^{S}_{max}(EDD)$. Both b and c precede a in the FCFS

Table 3
EDD order.

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 4
FCFS 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

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.

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

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

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

Proof. Case ordering (lines 1 and 2) requires $O(n \log n)$ 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 $O(n^2)$. The recalculation of $T^S_\xi(\text{A1}')$ (line 10) and the reevaluation of Γ (line 11) require $O(n)$. Hence the overall complexity is $O(n^3)$.□

In terms of correctness, we first analyze the relationships between the sets Γ and Ψ in Algorithm A1. We show that $\Gamma \neq \varnothing \rightarrow \Psi \neq \varnothing$, 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. $C^S_{(j)}(\text{A1}') \leq C^S_{(j)}(EDD)$ *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 $\Omega_{(i),(j)}(\text{EDD})$, i.e., $\Omega_{(i),(j)}(\text{A1}') \setminus \Omega_{(i),(j)}(\text{EDD}) \neq \varnothing$. 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 $\xi_{min} = \arg\min_{\xi \in \Omega_{(i),(j)}(\text{A1}')}\{d_\xi\}$ 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:

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

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

Example 1 nicely illustrates the property 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 also throughout the algorithm execution. The completion times of the cases that bypassed case c with respect to the EDD order were 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′, were 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:

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

Table 5
A1 final sequence.

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

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 $\Gamma \neq \varnothing \rightarrow \Psi \neq \varnothing$.

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 $\forall \xi \in J$:

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

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:

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

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

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

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 $C^{S'}_\xi \leq t$ and is subtracted from $\Upsilon^{S'}(t)$ if $C^S_\xi \leq t$ and vice versa. Note that $\forall t: \Upsilon^S(t) \cap \Upsilon^{S'}(t) = \varnothing$ 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.

Algorithm 2. Scheduling with Completed Cases by S' (A2).

1: $t = 0$; $\Upsilon^S(t) = \Upsilon^{S'}(t) = \varnothing$; {Initialization}
2: $\sigma =$ Order cases in an INIT order
3: Calculate $T^S_{\max}(\text{INIT})$
4: Calculate $F^S_{\max}(\text{INIT})$
5: **while** $\sigma \neq \varnothing$ **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** $C^S_{\sigma(k)} - d_{\sigma(k)} \leq T^S_{\max}(\text{INIT})$ and $C^S_{\sigma(k)} - r_{\sigma(k)} \leq F^S_{\max}(\text{INIT})$ **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: $\tilde{t} = \max\{t, r_{\sigma(s)}\} + p_{\sigma(s)}$
34: $\Upsilon^S(\tilde{t}) = \{\Upsilon^S(t) \cup \sigma(s)\} \backslash \{\Upsilon^{S'}(t) \cup \Phi(t, \tilde{t})\}$
35: $\Upsilon^{S'}(\tilde{t}) = \{\Upsilon^{S'}(t) \cup \Phi(t, \tilde{t})\} \backslash \{\Upsilon^S(t) \cup \sigma(s)\}$
36: $\sigma = \sigma \backslash \sigma(s)$
37: $t = \tilde{t}$
38: **end while**

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 a priori) are shown in Table 6.

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

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

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:

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

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 $k\le j$ and set $C_d^S=\max\{12,6\}+1+2=15$ (line 13). Both conditions of line 17: $C_d^S-d_d=2\le T^S_{\max}(\text{INIT})=3$ and $C_d^S-r_d=9\le F^S_{\max}(\text{INIT})=11$ 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.

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, $p_{(j)}(j-i)\le 0$ represents an 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* $O(n^2)$, *A2 runs in* $O(n^3)$.

Table 6
Completion times of S'.

ξ	b	a	c	d	e
$C^{S'}_\xi$	6	4	13	12	7

Table 7
Outcome of Algorithm A2.

ξ	b	a	c	d	e
p^S_ξ	5	3	4	1	2
r_ξ	0	2	1	7	6
d_ξ	9	5	10	14	13
C^S_ξ	5	8	12	13	15
$C^{S'}_\xi$	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

Proof. Lines 2–4 require $O(n)$ time. The while loop repeats $O(n)$ times. Within this loop, the indices j (outer) and k (inner) each takes $O(n)$ times. Note that the summation in line 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 $O(n^2)$. Thus, A2 runs in $O(n^3)$.□

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 fits 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 global 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 synchronization 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 optimization 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 s, 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. Fig. 3a illustrates the structure and deterministic routing obtained from the schedule, which is plain sequence of resource servers. Fig. 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 synchronization 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*: Fig. 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 min. 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 min). 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 Fig. 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.

Fig. 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 min on average. This is an S2D comparison, and the result of hypothesis testing for the median of processing time being 30 min is not rejected with a significance level of 5%. Therefore, the duration of the service does not explain the operational deviation identified above.

Turning to the service policy, we realized 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

[2] http://ie.technion.ac.il/labs/serveng/
[3] https://en.wikipedia.org/wiki/Dagum_distribution

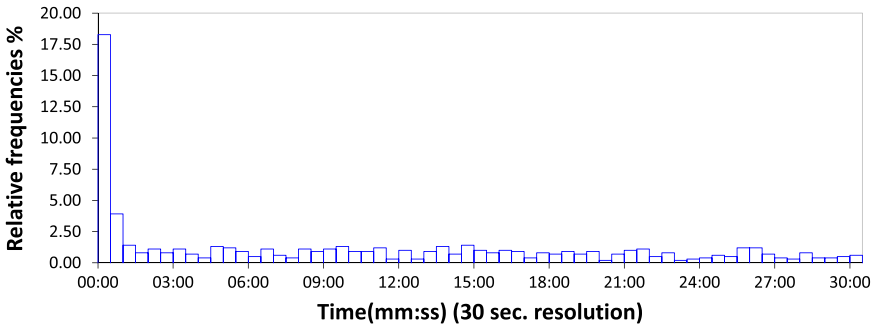


Fig. 5. Waiting time for Infusion (after vitals); Sample size=996, Mean=25 min, Stdev=29 min.

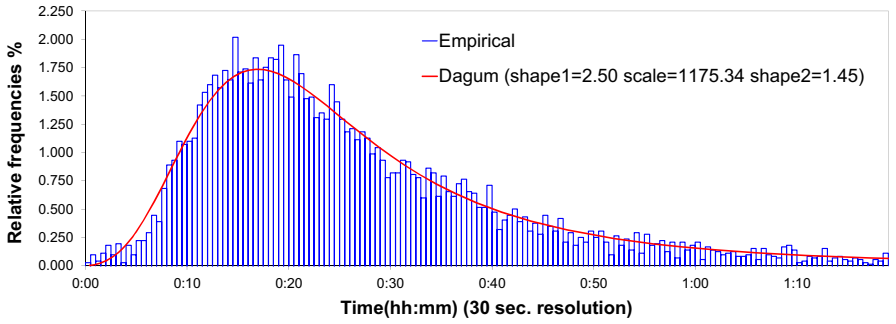


Fig. 6. Medication production time; Sample size=7187, Mean=30 min, Stdev=24 min.

(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.

Fig. 7 presents the estimated proportion of compliance to policy, as a function of 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 Algorithms 1 and 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 Fig. 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

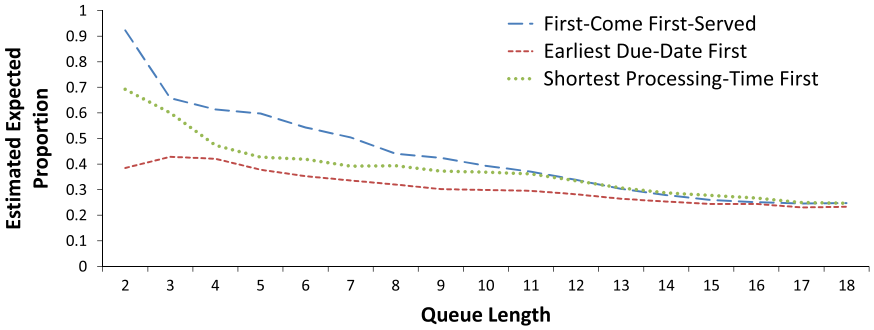


Fig. 7. Policy comparison for the pharmacy resource.

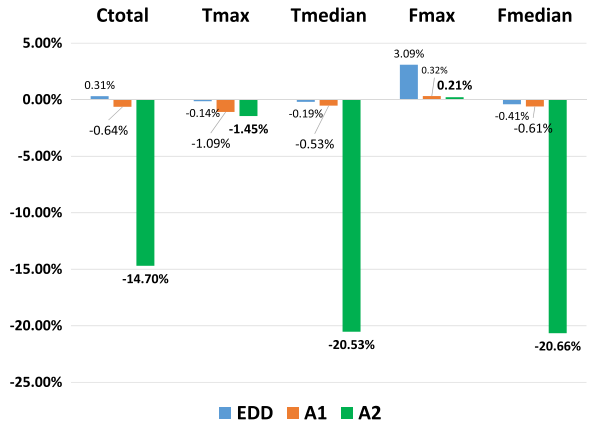


Fig. 8. Process improvement results: all patients.

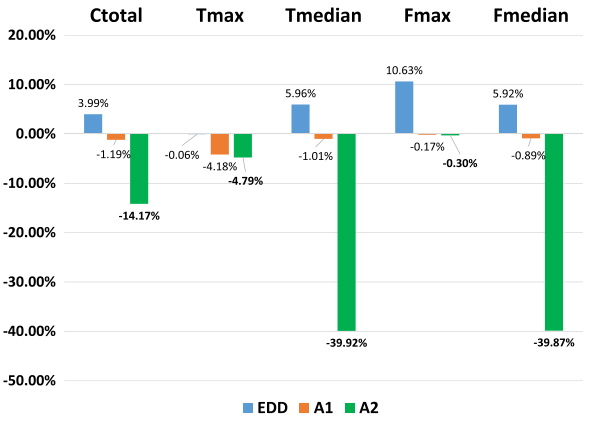


Fig. 9. Process improvement results: hematology patients.

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: Fig. 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 Algorithms 1 and 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, Fig. 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 stems from higher conceptual conformance of the single-server assumption for these patients.

Finally, Fig. 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 (Fig. 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 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 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

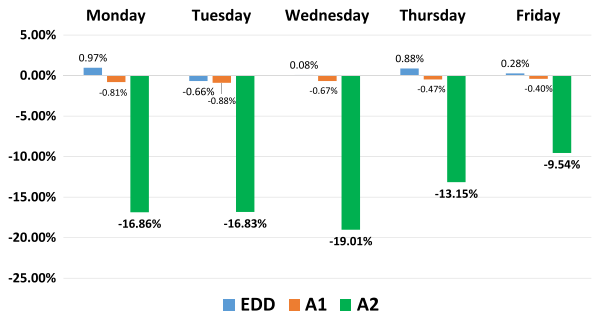


Fig. 10. Process improvement results for sum of completions: weekdays.

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 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 optimization 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 limitations, 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 Fig. 9, where conceptual conformance to these assumptions yields double the improvement in performance.

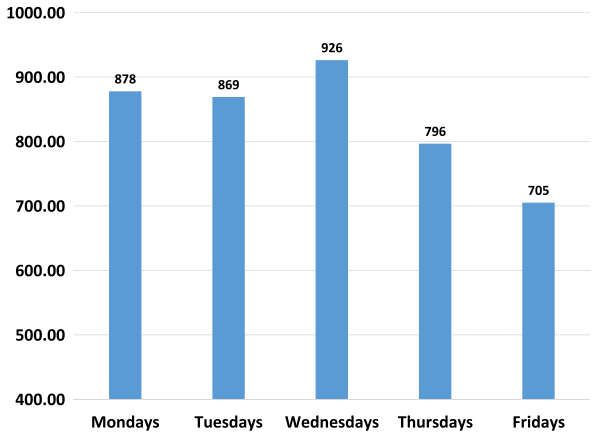


Fig. 11. Number of patients per day: weekdays average.

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, cf. [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, Chapter 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 [49–52]. 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 measure, 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, Berlin, 2013.
[2] M.S. Daskin, Service Science, Wiley.com, Hoboken, New Jersey, 2011.
[3] C.M. Froehle, M.J. Magazine, Improving scheduling and flow in complex outpatient clinics, in: Handbook of Healthcare Operations Management, Springer, Berlin, 2013, pp. 229–250.
[4] A. Gal, A. Mandelbaum, F. Schnitzler, A. Senderovich, M. Weidlich, Traveling Time Prediction in Scheduled Transportation with Journey Segments, Technical Report, Technion-Israel Institute of Technology, 2014.
[5] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining – predicting delays in service processes, in: 26th International Conference on Advanced Information Systems Engineering, CAiSE 2014, Thessaloniki, Greece, Proceedings, June 16–20, 2014, pp. 42–57.
[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, Hoboken, New Jersey, 2006.
[7] M.H. Ammar, S.B. Gershwin, Equivalence relations in queueing models of fork/join networks with blocking, Perform. Eval. 10 (3) (1989) 233–245.
[8] R. Atar, A. Mandelbaum, A. Zviran, Control of fork–join networks in heavy traffic, in: 2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Allerton Park & Retreat CenterMonticello, IL, USA, IEEE, 2012, pp. 823–830.
[9] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, S. Kadish, C.A. Bunnell, Discovery and validation of queueing networks in scheduled processes, in: 27th International Conference on Advanced Information Systems Engineering, CAiSE 2015, Stockholm, Sweden, Proceedings, June 8–12, 2015, pp. 417–433.
[10] G. Balbo, S. Bruell, M. Sereno, On the relations between bcmp queueing networks and product form solution stochastic petri nets, Petri Nets and Performance Models, IEEE, 2003, p. 103, http://doi.ieeecomputersociety.org/10.1109/PNPM.2003.1231547.
[11] F. Baccelli, W.A. Massey, D. Towsley, Acyclic fork–join queuing networks, J. ACM 36 (3) (1989) 615–642.
[12] F.P. Kelly, Networks of queues with customers of different types, J. Appl. Probab. (1975) 542–554.
[13] P.S. Adler, A. Mandelbaum, V. Nguyen, E. Schwerer, From project to process management: an empirically-based framework for analyzing product development time, Manag. Sci. 41 (3) (1995) 458–484.
[14] J.M. Harrison, Stochastic networks and activity analysis, Transl. Am. Math. Soc. Ser. 2 (207) (2002) 53–76.
[15] D.G. Kendall, Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov Chain, Ann. Math. Stat. 24 (3) (1953) 338–354.
[16] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, L. Zhao, Statistical analysis of a telephone call center, J. Am. Stat. Assoc. 100 (469) (2005) 36–50.
[17] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, Berlin, 2011.
[18] A. Burattin, Heuristics miner for time interval, in: Process Mining Techniques in Business Environments, Springer, Berlin, 2015, pp. 85–95.
[19] J.F. Allen, P.J. Hayes, A common-sense theory of time, in: IJCAI, vol. 85, 1985, pp. 528–531.
[20] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining—predicting delays in service processes, in: 26th International Conference on Advanced Information Systems Engineering, CAiSE 2014, Thessaloniki, Greece, Proceedings, June 16–20, 2014, pp. 42–57.
[21] P. Zhang, N. Serban, Discovery, visualization and performance analysis of enterprise workflow, Comput. Stat. Data Anal. 51 (5) (2007) 2670–2687.
[22] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Mining resource scheduling protocols, in: S.W. Sadiq, P. Soffer, H. Völzer (Eds.), 12th International Conference on Business Process Management, BPM 2014, Lecture Notes in Computer Science, Haifa, Israel, Proceedings, vol. 8659, Springer, Berlin, September 7–11, 2014, pp. 200–216.
[23] R.G. Sargent, Verification and validation of simulation models, in: S. Jain, Roy R Creasy, Jr., J. Himmelspach, K.P. White, M.C. Fu (Eds.), Winter Simulation Conference, WSC, 2011, pp. 183–198.
[24] P.J. Bickel, K.A. Doksum, Mathematical Statistics: Basic Ideas and Selected Topics, volume I, vol. 117, CRC Press, Boca Raton, Florida, 2015.
[25] T.W. Anderson, L.A. Goodman, Statistical inference about Markov Chains, Ann. Math. Statist. 28 (1) (1957) 89–110.
[26] P. Momcilovic, N. Trichakis, A. Mandelbaum, S. Kadish, C.A. Bunnell, Data-driven appointment scheduling under uncertainty, Working Paper.
[27] J.D. Gibbons, S. Chakraborti, Nonparametric Statistical Inference, Springer, Berlin, 2011.
[28] S. Maman, Uncertainty in the demand for service: the case of call centers and emergency departments (Master's thesis), 2009.
[29] J.D. Little, A proof for the queuing formula: L=λw, Oper. Res. 9 (3) (1961) 383–387.
[30] S.-H. Kim, W. Whitt, Estimating waiting times with the time-varying Little's law, Prob. Eng. Inf. Sci. 27 (4) (2013) 471–506.
[31] D.H. Maister, The Psychology of Waiting Lines, Harvard Business School, Boston, MA, USA, 1984.
[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, Telecommun. Syst. 21 (2–4) (2002) 149–171.
[33] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, Springer Science & Business Media, Berlin, 2012.
[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: 13th International Conference on Business Process Management, BPM 2015, Innsbruck, Austria, Proceedings, August 31–September 3, 2015, pp. 35–52.
[35] J.R. Jackson, Scheduling a Production Line to Minimize Maximum Tardiness, Technical Report, DTIC Document, 1955.
[36] J.K. Lenstra, A.R. Kan, P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.
[37] J. Du, J.Y.-T. Leung, Minimizing total tardiness on one machine is np-hard, Math. Oper. Res. 15 (3) (1990) 483–495.
[38] A. Farcomeni, A review of modern multiple hypothesis testing, with particular attention to the false discovery proportion, Stat. Methods Med. Res. 17 (4) (2008) 347–388, http://dx.doi.org/10.1177/0962280206079046.
[39] E. Arjas, T. Lehtonen, Approximating many server queues by means of single server queues, Math. Oper. Res. 3 (3) (1978) 205–223.
[40] S. Halfin, W. Whitt, Heavy-traffic limits for queues with many exponential servers, Oper. Res. 29 (3) (1981) 567–588.
[41] J.A. Buzacott, J.G. Shanthikumar, Stochastic Models of Manufacturing Systems, Prentice Hall, Englewood Cliffs, NJ, 1993.
[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.
[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] W.M. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Inf. Syst. 36 (2) (2011) 450–475.
[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, Lecture Notes in Computer Science, vol. 8274, Springer, Berlin, 2013, pp. 389–403.
[46] A. Rozinat, R. Mans, M. Song, W.M.P. van der Aalst, Discovering simulation models, Inf. Syst. 34 (3) (2009) 305–327.
[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, Berlin, 2014, pp. 564–579.
[48] J. Nakatumba, W.M. van der Aalst, Analyzing resource behavior using process mining, in: Business Process Management Workshops, Springer, Berlin, 2010, pp. 69–80.
[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.
[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.
[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.
[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.
[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.
[54] M. de Leoni, W.M.P. van der Aalst, B.F. van Dongen, Data- and resource-aware conformance checking of business processes, in: 15th International Conference on Business Information Systems, BIS 2012, Vilnius, Lithuania, May 21–23, 2012. Proceedings, 2012, pp. 48–59.
[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, Helsinki, Finland, 2011, pp. 55–64.
[56] M. Dumas, L. García-Bañuelos, Process mining reloaded: event structures as a unified representation of process models and event logs, in: 36th International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2015, Brussels, Belgium, June 21–26, 2015, Proceedings, 2015, pp. 33–48.
[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: 13th International Conference on Business Process Management, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, Proceedings, 2015, pp. 386–405.