Data-Driven Performance Analysis of Scheduled Processes

Arik Senderovich[1], Andreas Rogge-Solti[2], Avigdor Gal[1],
Jan Mendling[2], Avishai Mandelbaum[1], Sarah Kadish[3], and Craig A. Bunnell[3]

[1] Technion – Israel Institute of Technology
{sariks@tx,avigal@ie,avim@ie}.technion.ac.il
[2] Vienna University of Economics and Business
{andreas.rogge-solti,jan.mendling}@wu.ac.at
[3] Dana-Farber Cancer Institute
{sarah_kadish,craig_bunnell}@dfci.harvard.edu

Abstract. The performance of scheduled business processes is of central importance for services and manufacturing systems. However, current techniques for performance analysis do not take both queueing semantics and the process perspective into account. In this work, we address this gap by developing a novel method for utilizing rich process logs to analyze performance of scheduled processes. The proposed method combines simulation, queueing analytics, and statistical methods. At the heart of our approach is the discovery of an individual-case model from data, based on an extension of the Colored Petri Nets formalism. The resulting model can be simulated to answer performance queries, yet it is computational inefficient. To reduce the computational cost, the discovered model is *projected* into Queueing Networks, a formalism that enables efficient performance analytics. The projection is facilitated by a sequence of *folding* operations that alter the structure and dynamics of the Petri Net model. We evaluate the approach with a real-world dataset from Dana-Farber Cancer Institute, a large outpatient cancer hospital in the United States.

1 Introduction

Scheduled processes are pervasive in our lives. In services, manufacturing systems and transportation, one encounters corresponding schedules, such as appointment books, production plans and bus timetables. Typically, it is of central importance for companies to analyze the performance of their processes. Data stemming from event logs of these processes play an increasingly important role in this context [1] and first contributions have been made to investigate scheduled process from a conformance perspective, which is grounded in process mining concepts [2].

While there are powerful methods for performance analysis in prior research, these are bound to different types of limitations. First, analytical work in the area of operations research often does not provide direct techniques to make use of available execution data [3]. Second, process mining methods for performance analysis carry their own limitations. In particular, Petri Net-based techniques discover fine-grained models to capture the process perspective at the individual-case level [4, 5], but these analyses

are restricted to simulation. Third, queue mining [6] focuses on the resource perspective to efficiently answer performance questions (*e.g.*, delay prediction), while ignoring other aspects of the underlying process, such as the control-flow perspective.

The main purpose of this paper is to provide a flexible, efficient and accurate method for data-driven performance analysis of scheduled processes. We achieve this goal by bridging the gap between the discovery of Petri Net simulation models and queue mining. Our approach consists of three steps: discovery, folding, and projection. At the foundation of the approach lies a discovery procedure that utilizes the schedules and execution logs of the underlying process to construct and enrich a novel type of Colored Petri Nets, the Queue-Enabling Colored Stochastic Petri Nets (QCSPN). The proposed formalism is highly expressive and includes stochastic times, scheduling mechanisms and queues.

To reduce the computational effort required to simulate the resulting model, we *project* the QCSPN into a performance-oriented formalism, Queueing Networks. Several types of Queueing Networks and their approximations are well-known for their complexity-reducing characteristics [7]. However, there is an expressiveness gap between Petri Nets and Queueing Networks, which does not allow for an immediate transformation of one formalism into the other [8]. To close the gap between the two formalisms, we introduce the concept of *folding* that alters the structure and dynamics of the originating Petri Net, thus making the Petri Net projection-ready. We test our approach by conducting a predictive evaluation against a real-world dataset of the Dana-Farber Cancer Institute, a large outpatient cancer hospital in the United States. Our experiments demonstrate the influence of abstraction on prediction accuracy, depending on the correctness of folding assumptions. Moreover we show that projection of Petri Net models into Queueing Networks improves accuracy, while benefiting from run-time efficiency.

This paper is structured as follows. Section 2 presents an overview of our approach and a running example. Section 3 presents our data model and the Queue-Enabling Colored Stochastic Petri Nets. Section 4 defines the discovery algorithm for Queue-Enabling CSPNs (QCSPN). Section 5 formalizes the techniques for folding and projecting QCSPNs. Section 6 describes an empirical evaluation of our approach. Section 7 discusses related work before Section 8 concludes.

2 Approach Overview

This section describes the need for our approach. To this end, we refer to a use-case that is inspired by data from the Dana-Farber Cancer Institute. The outpatient hospital is equipped with a Real-Time Location System (RTLS) that tracks, via 905 sensors, approximately 900 patients per day. These patients are served by 300 healthcare providers (*e.g.*, physicians, nurse practitioners, registered nurses) supported by 70 administrative staff, and occupying 7 floors.

The schedule of an ambulatory patient typically includes a blood draw, an examination by a physician or nurse practitioner, and a chemotherapy infusion (Figure 1). The process may vary among cases, with some patients skipping activities, while others

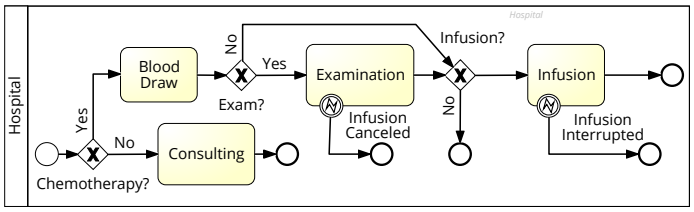

Fig. 1: The main process in Dana-Farber Cancer Institute.

having additional activities (*e.g.*, acupuncture therapy, speech therapy, radiology scans).

Example 1 *Consider two specific patients, pat1 and pat2, and their scheduled routes for the same day. Both patients are to go through a* blood-draw *activity that is scheduled to be performed by Registered Nurse Tanya (we write RNTanya, as an abbreviation). Then, pat1 is to go through an* examination *activity, performed by two physicians (medical doctors): MDVictor and MDElaine. Pat2 is also scheduled to go through an* examination*, which includes a speech therapy appointment as a parallel activity; the examination is planned to be carried out by MDElaine, while the speech therapy will be performed by Speech Therapist Brooke (STBrooke).*

Performance questions arise from several perspectives. From the patient's perspective, it is important to predict their length-of-stay to reduce uncertainty about the remainder of their day. From the hospital's perspective, assessing the utilization profiles of resources is a key issue. These questions can be answered either off-line (*e.g.*, the day before) or in real-time. For the off-line scenario, a data-driven simulation model that captures every phase of the process can be invaluable (because run-time is not an issue). This detailed case-level view is not covered by the queueing perspective. For real-time analysis, an efficient and relatively accurate model with a fast response time is required. This cannot be achieved by simulation, since its run-time may be slower than the required response time. Our approach offers methods to move freely in the spectrum between detailed-complex models and abstract-efficient models.

Figure 2 presents the outline of our approach with section numbers being on the arcs. The phases of our approach are depicted by rectangles, modeling formalisms (QCSPN, Queueing Networks) are shown by circles, while the resulting models (after each phase) are shown above the relevant phases. The approach starts with a data log, which contains details on both the scheduled tasks and the corresponding actual execution times. As our formalism, we adjust Colored Petri Nets [9] to form Queue-Enabling Colored Stochastic Petri Nets (QCSPN) with time distributions, scheduling transitions and queueing stations. In the first phase, the data is used to discover a simulation-ready QCSPN model that represents schedule in detail. The main drawback of the resulting model is that one 'cannot see the forest for the trees,' meaning that the amount of details cause the QCSPN model to be less effective in terms of run-time complexity.

To resolve this inefficiency, we propose the second and third phases, *folding* and *projection*. An abstracted version of the original QCSPN model is produced by applying

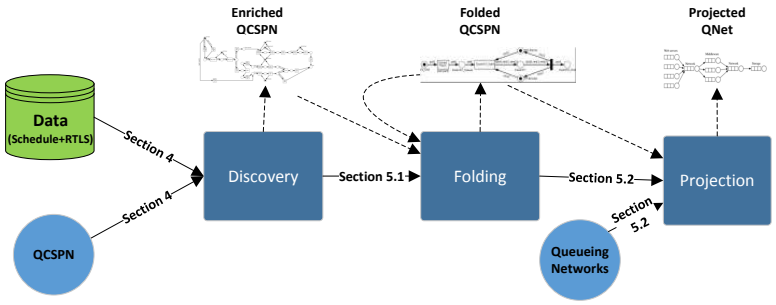

Fig. 2: An outline of our approach.

a sequence of foldings, which alter the model at the net-level. This phase bridges the expressiveness gap between Petri Nets and Queueing Networks (QNets), thus allowing for the last phase of our approach, which is projection of the QCSPN into Queueing Networks.

3 Models

We introduce two models. A *schedule log*, which serves as the data model, and *Colored Stochastic Petri Nets*, which is a modeling formalism that is based on Coloured Petri Nets with stochastic delays and scheduling transitions. For the latter, we define Queue-Enabling CSPNs, to be used to construct projection functions from Petri Nets into Queueing Networks.

Data Model. A *schedule log* contains a set of tasks and the actual execution times of these tasks. A *task* is defined as a relation between cases, activities, resources, and times. Let Θ be the universe of tasks, A be the domain of activities, R the set of resources, $\mathbb{TS}$ be the set of timestamps (Unix time) and I the set of case identifiers. Then, the task information is defined as follows.

Definition 1 (Task Information) Task information $\mathcal{I} = \langle \xi, \alpha, \rho, \tau, \delta, \tau_{start}, \tau_{end} \rangle$ *is a tuple satisfying the following requirements:*

- $\xi : \Theta \to I$ *assigns a case identifier to a task.*
- $\alpha : \Theta \to A$ *assigns an activity to a task.*
- $\rho : \Theta \to 2^R$ *assigns a set of resources to a task.*
- $\tau : \Theta \to \mathbb{TS}$ *assigns a timestamp representing the planned start time to a task.*
- $\delta : \Theta \to \mathbb{N}^+$ *assigns a scheduled duration to a task.*
- $\tau_{start} : \Theta \to \mathbb{TS}$ *assigns the observed start time to a scheduled task.*
- $\tau_{end} : \Theta \to \mathbb{TS}$ *assigns the observed end time to a scheduled task.*

Given task information we define schedule logs as follows.

Definition 2 (Schedule Log) *Let $\Theta_P \subseteq \Theta$ be a set of scheduled tasks. The schedule log is defined as a tuple $\langle \Theta_P, \mathcal{I} \rangle$, which contains all* scheduled tasks *and* task information.

Table 1: Schedule log for the running example.

Case	Activity	Resources	Scheduled Start	Scheduled Duration	Actual Start	Actual End
pat1	Blood-Draw	[RNTanya]	9:00AM	10 (MIN)	9:05AM	9:10AM
pat1	Exam	[MDVictor, MDElaine]	10:00AM	30 (MIN)	9:55AM	10:20AM
pat2	Blood-Draw	[RNTanya]	9:10AM	15 (MIN)	9:15AM	9:27AM
pat2	Exam	[MDElaine]	9:40AM	20 (MIN)	9:30AM	9:45AM
pat2	Speech-Therapy	[STBrooke]	9:40AM	50 (MIN)	9:35AM	10:32AM

Table 1 shows a schedule log for the running example. Notice that the scheduled times and actual times do not necessarily match.

Formalism. As our formalism, we build upon the Colored Petri Nets (CPN) by Jensen [9] to discover, enrich and simulate the scheduled process. To this end, we extend the CPN model with *scheduling transitions* and *distribution functions* of firing delays. Below, we define the structure of the CSPN formalism and specify its state and dynamics (marking and firing semantics, respectively).

Definition 3 (CSPN Structure) *The* structure *of a CSPN is a tuple* $N = \langle \mathcal{C}, P, T, F, N, \mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{S} \rangle$ *where:*

- $\mathcal{C} = \Sigma \times \mathbb{TS}$ *is the finite set of types* Σ*, called* color sets*, and the associated timestamps.*
- P *is a finite set of* places*.*
- $T = T_R \cup T_S$ *is a finite set of* transitions*, such that* T_R *is the 'regular' timed transitions, and* T_S *are referred to as 'scheduling' transitions.*
- F *is a finite set of* arcs *representing flow such that:* $P \cap T = P \cap F = T \cap F = \emptyset$*.*
- $N : F \to P \times T \cup T \times P$ *is a* node *function.*
- $\mathcal{G} : T \to Expr$ *is a* guard *function that evaluates to boolean predicates.*
- $\mathcal{E} : F \to Expr$ *is an* arc expression *function that evaluates to a set of types.*
- $\mathcal{D} : T_R \to (\mathbb{N}^+ \to [0, 1])$ *are distribution functions of firing delays in seconds that are associated with timed transitions,*
- $\mathcal{S} : T_S \to \mathbb{TS}$ *are timestamps assigned to scheduling transitions,*

In the remainder of the paper, we adopt the common Petri net bullet notation for in-places and out-places of transitions. That is, the in-places $\bullet t$ of a transition t are $\{p \in P \mid (p, t) \in F\}$, and the out-places $t\bullet$ are $\{p \in P \mid (t, p) \in F\}$.

Figure 3 demonstrates parts of the formalism by showing the CSPN that corresponds to the blood draw task for the first patient *pat1* in our running example. We closely follow the semantics as introduced by Jensen for CPNs [9]. The arc expressions contain variables that can be bound to typed tokens. For example, the variable *pat* can be bound to patient *pat1*. A transition t is *color enabled* in a binding, if the input places $\bullet t$ contain tokens that satisfy the arc expressions and the guard function $\mathcal{G}(t)$ evaluates to true given the binding. The binding by the arc expressions takes care of proper routing of the tokens to their respective output places. In our example, we require the token of *RNTanya* to return to its place so that she can draw blood from the next patient. This is taken care of by the variable *res* in the corresponding arc expressions.

Besides the colors, a token carries an associated time that specifies at which point in time the token becomes *ready* for the next firing. This depends on the global clock

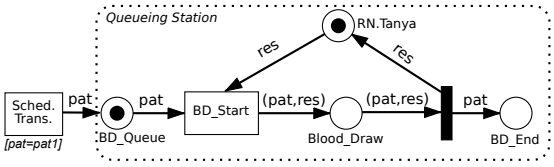

Fig. 3: An illustration of the blood-draw task for pat1; the elements of the queueing station are highlighted with the dashed box.

g, i.e., the token time must be smaller or equal to g. Transitions are eager to fire, that is, whenever a transition is color enabled and the tokens are ready, it immediately fires. The timestamp ts of each of the produced tokens is set to $g + \Delta(t, g)$, where $\Delta(t, g)$ is the firing delay of t at time g:

$$\Delta(t,g) = \begin{cases} d & \text{if } t \in T_R, \\ \max(g, \mathcal{S}(t)) - g & \text{if } t \in T_S, \end{cases} \tag{1}$$

Here, d is a realization of the random duration D_t that comes from distribution $\mathcal{D}(t)$. We call a transition t_i with all the probability mass of $\mathcal{D}(t_i)$ on 0 an *immediate* transition an depict it with a bar in the model, as known from the GSPN formalism [10].

Queue-Enabling CSPN. In this part we define the *Queue-Enabling CSPN (QC-SPN)*, which is a CSPN with scheduling transitions, queueing stations, and fork/join constructs.

Definition 4 (Queueing Station) *A* queueing station *is a CSPN, where*

- $P = \{p_q, p_a, p_e, p_{r_1}, ..., p_{r_K}\}$*, with* p_q *being a queueing place,* p_a *being the on-going activity place,* p_e *being the end place and* $K \in \mathbb{N}$ *being the number of service providing resources per station,*
- $T = \{t_s, t_e\}$ *being the start and end transitions,*
- $F = \{f_{in_i}, f_{serve}, f_{served}, f_{out_i}, f_{leave}\}$ *are the flow arcs with* $i = 1, ..., K$ *and,*
- $N(f_{enter}) = (p_q, t_s)$*,* $N(f_{in_i}) = (p_{r_i}, t_s)$*,* $N(f_{serve}) = (t_s, p_a)$*,* $N(f_{served}) = (p_a, t_e)$*,* $N(f_{out_i}) = (t_e, p_{r_i})$*,* $N(f_{leave}) = (t_e, p_e)$*.*

For example, in Figure 3, the subnet that starts with the queueing place 'BD_Queue' and ends at the 'BD_End' place, is a queueing station. We are now ready to define the Queue-Enabling CSPN (QCSPN).

Definition 5 (Queue-Enabling CSPN (QCSPN)) *A Colored Stochastic Petri Net* $\langle \mathcal{C}, P, T, F, N, \mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{S} \rangle$ *is called* Queue-Enabling*, if the CSPN contains a single source* p_α*, a single sink* p_ω*, and every other node (*$n \in P \cup T \setminus \{p_\alpha, p_\omega\}$*) of the CSPN belongs to either,*

- *A queueing station, or*
- *An immediate split or join transition* t_i *(with* $| \bullet t_i | > 1 \wedge |t_i \bullet | = 1$ *or* $| \bullet t_i | = 1 \wedge |t_i \bullet | > 1$*), or*
- *There exist a scheduling transition* $t_\Sigma \in T_S$*, such that* n *is the predecessor place of the scheduling transition or* t_Σ *itself.*

4 Discovery of Queue-Enabling CSPN Models

This section is devoted to the discovery and enrichment of Queue-Enabling Colored Stochastic Petri Nets (QCSPN) from the schedule log. To discover the QCSPN, we extend the approach of van der Aalst's work on scheduling with Petri Nets [11] to include scheduling transitions and stochastic times. First, we provide an overview of preprocessing and assumptions required for discovering and enriching the QCSPN model. Then, we demonstrate a three-step discovery algorithm that constructs the QCSPN. Finally, an enrichment procedure of the model from data is described.

Preprocessing and Assumptions. Precedence constraints (synchronization points) are a key feature of scheduled processes, ensuring that cases are not allowed to continue to a new task before a subset of other tasks is performed. To handle parallelism, we apply a *preprocessing* phase in which we detect parallel tasks using interval calculus [12]. Thereby, we assume tasks to be parallel if the intersection of their planned times is not empty. Henceforth, we shall assume the existence of a *parallelism set*, $\Pi \subseteq 2^{\Theta}$, which contains *sets of tasks* that are scheduled to be performed in parallel. The set Π is a partition of Θ_P, since we assume transitivity of the parallelism property, thus avoiding new splits prior to joining previous splits. Three more assumptions are used in the discovery process, as follows:

- **Work conserving:** Resources become available immediately after the completion of a task.
- **Temporal deviations:** The scheduled tasks may deviate in time only (no activities, resources or routing deviation).
- **Duration dependencies:** Activity durations depend only on the activity and its planned duration (independent of marking components, i.e. case identifier, resources, and scheduled time).

Discovery. The discovery algorithm comprises three steps, construction of queueing stations, synchronization of parallel tasks, and initialization of the state (marking and global clock). Next, we go over the steps and relate them to the proposed models (Section 3).

Step 1: Construct Queueing Stations. We start by inserting all resource places, $\{p_r \mid r \in R\}$. Then, for each task $\theta \in \Theta_P$ of the schedule log, a corresponding queueing station is created as follows. The activity place is defined as $p^{\theta}_{a}, a = \alpha(\theta)$. The resource places that are connected to the starting transitions are $\{p_r \mid r \in \rho(\theta)\}$; the durations of timed transitions t^{θ}_{s} are set to be deterministic (*i.e.*, according to plan) with $D_{t^{\theta}_{s}} = \delta(\theta)$. Arcs that connect places and transitions receive arc-expressions, which verify that resource tokens and case tokens are separated and routed appropriately. Subsequently, scheduling transitions are inserted to precede queueing places, p^{θ}_{q}, to prevent an ahead-of-time arrival into the queueing station. Every scheduling transition $t^{\theta}_{\Sigma} \in T_S$ is assigned with a timestamp $\mathcal{S}(t^{\theta}_{\Sigma}) = \tau(\theta)$ according to the earliest start time of the activity. Finally, we add a source and sink place, p_α and p_ω, respectively.

Step 2: Synchronize Parallel Constructs. In this step, we add split and join transitions for every parallelism class in Π. Let $\pi \in \Pi$ be a set of parallel tasks, with $|\pi| > 1$ (parallelism classes may be singletons for sequential tasks). We add a *split* transition t^{π}_{sp} to the set of transitions and connect it to each scheduling transition $t^{\theta}_{\Sigma}, \theta \in \pi$ via a new scheduling place. Then, we add *join* transitions t^{π}_{j} after each parallel construct

to express the synchronization of the concurrent tasks. Each of the join transitions is assigned with a guard that verifies that joining is performed only for tokens with the same static component, *i.e.*, case identifier. Figure 4 demonstrates a parallel construct for *pat2* from our running example. According to schedule, the patient is to undergo examination and speech therapy in an overlapping period of time.

Step 3: Initialize State. This step sets the initial marking and the global clock. The global clock, which is the dynamic part of the state, is set to zero. For the initial marking, all case tokens start at p_α, while resource tokens reside in their corresponding places. The number of resource tokens in each resource place corresponds to the offered capacity of that resource, which is the maximum number of tasks that a single resource is scheduled to perform at the same time. For our running example, this allows for a nurse to attend multiple infusion patients in parallel. The static marking component of case and resource tokens is their unique identifiers. The timestamp component for case tokens is initialized to be zero. The timestamp component of resource tokens is initialized to the timestamp of the first scheduled task for the corresponding resource.

Enrichment. Once the QCSPN model is discovered from the schedule, we enrich it based on actual execution times per task by replacing deterministic durations with stochastic ones. To this end, we apply the techniques for enhancement of non-Markovian Stochastic Petri Nets with non-parametric kernel density regression [5]. Other model components that are often stochastic, *e.g.*, exception-handling mechanisms and routing, are assumed to be driven by case-information and therefore deterministic. The outcome of the discovery and enrichment steps is a simulation-ready QCSPN model, which we refer to as N_0.

5 Folding and Projection of QCSPN into Queueing Networks

In this section we introduce the concepts of *folding* and *projection* of QCSPNs. First, we define the folding function and provide with several examples of foldings. Then, we define athe projection function and demonstrate a single projection with the help of a sequence of foldings.

5.1 Folding of QCSPN.

Let M_{QCSPN} be the universe of all QCSPN models and let $\mathbb{A}$ be the universe of possible model assumptions, (*e.g.*, activity times are exponentially distributed).

Definition 6 (Folding Function) *A* folding function $\psi_{\mathcal{A}} : M_{QCSPN} \rightarrow M_{QCSPN}$, *creates a new QCSPN model, under a set of assumptions* $\mathcal{A} \subseteq \mathbb{A}$.

Below, we provide several examples of folding functions. For each function, we explain the net-level changes that it requires, and demonstrate it with our running example. We omit the formal proofs that show that the resulting nets are QCSPN, due to space restrictions.

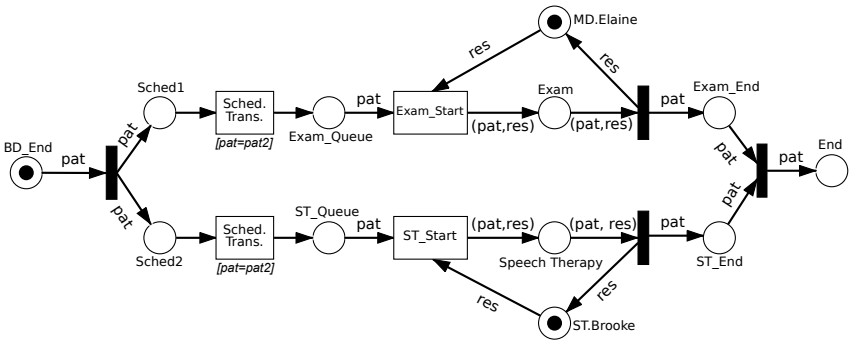

Fig. 4: Parallel examination and speech therapy for pat2.

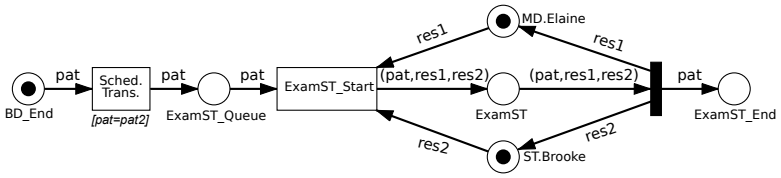

Fig. 5: Folding function: Remove parallelism.

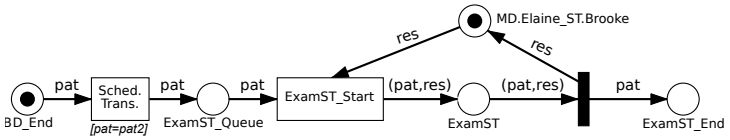

Fig. 6: Folding function: Remove shared resources.

Folding 1: Remove Parallelism (RP) Parallelism is well-known for its negative influence on the analytical tractability of Queueing Networks [13]. This motivates us to consider a folding operation ψ_{RP} that transforms N_0 into a concurrency-free model, $\psi_{RP}(N_0)$. Specifically, ψ_{RP} adds the assumption: "all parallel tasks must start and end at the same time occupying all resources that were scheduled to perform the (originally) parallel tasks."

Without loss of generality, we show the net-level changes that the RP function implies on a single parallel class of tasks, $\pi \in \Pi$. Note that the marking-related elements remain unchanged. For every $\theta \in \pi, |\pi| > 1$, the folding function removes the corresponding queueing station (non-resource places, transitions, flow relation). Moreover, the corresponding split and join transitions $(t^{\pi}_{split}, t^{\pi}_{join})$ are also removed from the net.

Subsequently, a single queueing station that corresponds to a new task θ' is created and is connected to all resources places that were connected to the original tasks $\theta \in \pi$. The activity name for the new station is defined as a concatenation (denoted $\biguplus$) of all previously parallel activities, *i.e.*, $\alpha(\theta') = \biguplus_{\theta \in \pi} \alpha(\theta)$. The random duration of the timed transition t' that corresponds to the new activity is written as $D_{t'} = \max_{\theta \in \pi}[D_{t^{\theta}_s}]$, with

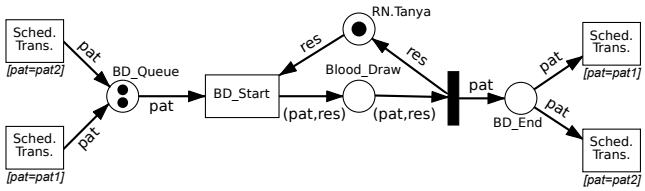

Fig. 7: The result of fusing queues

$D_{t_s^\theta}$ being the random duration of task θ. The scheduling transitions per parallel branch are folded into a single scheduling transition $t_\Sigma^{\theta'}$ with $\mathcal{S}(t_\Sigma^{\theta'}) = \min_{\theta \in \pi}[\mathcal{S}(t_\Sigma^{\theta})]$, i.e. the scheduled start time of the new task is the earliest among all start times of the original parallel tasks. Figure 5 shows $\psi_{RP}(N_0)$ for the running example. We observe that for *pat2*, a new task 'Exam_SpeechTherapy' that requires both resources is created. The synchronization constructs for the previous two queueing stations no longer exist.

Folding 2: Remove Shared Resources (RSR) Shared resource possession imposes mathematical difficulties when analyzing queueing systems [14]. Therefore, as a next step toward projection of N_0 into Queueing Networks we apply folding function ψ_{RSR}, which removes shared resources. The underlying assumption for the RSR function is the following: "resources $R' \subseteq R$ that share task θ can be combined into a single resource with $\rho(\theta) = \{ \biguplus_{r \in R'} r \}$". In other words, the set of resources R' becomes a new resource that is added to R. Figure 6 demonstrates the result of $\psi_{RSR}(\psi_{RP}(N_0))$: a new resource *MDElaine_STBrooke* is created for the second task of patient *pat2*.

Function 3: Fuse Queues (FQ) In this step, we further reduce model complexity by fusing queueing stations that perform the same activities and share the same resources. For example, consider two queueing stations that correspond to two tasks, θ_1 and θ_2, such that $\alpha(\theta_1) = \alpha(\theta_2)$ and $\rho(\theta_1) = \rho(\theta_2)$. The fusion merges the queuing, service, and end places, as well as the start and end transitions for the two tasks. Duplicate arcs are removed, as the arc-expressions are equal. Scheduling transitions are not fused and govern the routing of the corresponding patients through the fused queueing stations.

Figure 7 presents $\psi_{FQ}(N_0)$ for $RNTanya$'s two blood draw tasks of our running example. We observe that in the new net, two case tokens can reside in the queueing place and wait for $RNTanya$. Here, if we assume that both patient tokens are *ready*, the conflict between them needs to be resolved. We assume that cases are served according to the earliest-due-data first (EDD) policy, i.e. the case token with the smaller timestamp will get served first. The fuse queues folding does not change the performance characteristics of the QCSPN model. Nevertheless, it is a required step that enables projection into Queueing Networks, since it joins the otherwise scattered queues of activity-resource pairs.

Function 4: Remove Scheduling Constraints (RSched) The last folding function builds on the assumption that "scheduling constraints are not enforced". In other words, cases

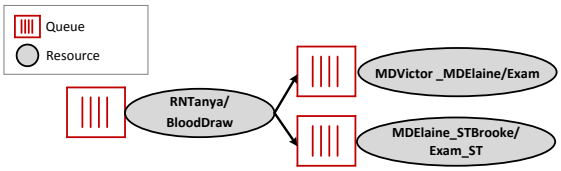

Fig. 8: The result of the projection function: Queueing Network with distinguishable cases.

that finish service in a certain queueing station are immediately routed to the succeeding station according to schedule. The RSched folding implies a very simple change at the net level: every scheduling transition is turned into an immediate transition with corresponding guards that verify the identity of cases for routing purposes.

5.2 Projection of QCSPN into Queueing Networks.

The intuition behind the idea of projecting QCSPN into Queueing Networks is straightforward. Queueing networks are directed graphs, with vertices being single-station queues and edges being the routing mechanism that communicate customers between these queues[4]. Therefore, as a first step of projection queueing stations of the originating QCSPN are transformed into vertices of the target Queueing Network. Every vertex of the Queueing Network can be characterized by the number of resources that reside in that vertex, which serve customers according to some service time distribution. These times correspond to the random durations of the timed transitions in the QCSPN. The matching between cases and resources are governed by service policies, *e.g.* first-come-first-served (FCFS) or earliest-due-date first (EDD). At service completion, customers are routed to the next queueing vertex (either deterministically or according to assigned probabilities). The described behavior corresponds in a one-to-one manner to our definition of a QCSPN, and provides the basis for the construction of a projection function. Formally, the projection function is defined as follows.

Definition 7 (Projection Function) *Let M_{QN} be the universe of all Queueing Networks. A projection function $\phi : M_{QCSPN} \rightarrow M_{QN}$ creates a Queueing Network from the originating QCSPN.*

Figure 8 presents a projection function, operated on the folded version of our running example. The folding includes the four foldings that we provided in Section 5.1, i.e. Remove Parallelism, Remove Shared Resources and Fuse Queues, and Remove Scheduling Constraints, in the order of their presentation. The resulting Queueing Network has distinguishable customers, single resource per-station and does not allow for exogenous arrivals (customers start in the system at the time of their arrival).

[4] Due to Queueing Network conventions we write the terms customers and cases interchangeably

6 Evaluation

In this section we describe the results of an empirical evaluation of the proposed approach. Here, we aim at demonstrating the usefulness of the method spectrum: from detailed Petri Net-based simulators, through folded versions of the originating model, to predictors that are based on the projected Queueing Network. We start with the main aspects that involve the implementation of our techniques. Then, we describe the dataset and the design of our experiment. We conclude the section with our main results and a discussion of the evaluation.

Implementation. We implemented the model construction, its enrichment with historical data, the simulation semantics, the folding operations and a projection into the Queueing Theory based snapshot predictor. The implementation uses the Python programming language and builds on the open-source SNAKES framework [15]. The implementation is available as a free open-source project[5].

Data Description. The data that we consider comes from the Dana-Farber Cancer Institute. We combine two datasets into one: the scheduled visits and their corresponding execution times. The former contains a detailed schedule for each day, while the latter is based on Real-Time Location System measurements of that day. Specifically, every patient's path is measured by the RTLS and matched to the originating appointments that reside in the patient's schedule. We utilize a year's worth of data, for year 2014 (222 regular workdays, nearly 140000 scheduled visits), for training the enrichment algorithms and testing our techniques. The training set includes 212 regular workdays, while the test set consists of 10 workdays, selected at random. We excluded special days (*e.g.*, Christmas) with irregularly high or low workloads from the random selection of the test days.

Experimental Setup. The design of our experiment is as follows. We predict the length-of-stay (i.e. the time in process from start to end) for every scheduled patient over the 10 test days. The prediction is then compared against the actual length-of-stay. Patients are assumed to arrive at their real arrival-time, as it is recorded in the data. The *uncontrolled* variables in our experiments are the root of the mean-squared prediction error (RMSE), and the mean error. The former measures the deviation between the predicted value and the actual value of the length-of-stay (LOS), while the latter shows the 'bias' of the predictors. The *controlled* variable is the model that we use for prediction of the LOS. The QCSPN models are discovered from the test day's schedules and are then enriched by the training set.

For prediction, we consider the following five models: the original model (N_0), the no-concurrency model ($\psi_{RP}(N_0)$), the removed shared resources model without concurrency ($\psi_{RP,RSR}(N_0)$), the scheduling transitions and fuse queues model ($\psi_{RP,RSR,FQ,RSched}(N_0)$), and the queueing predictor that corresponds to the projected model ($\phi(\psi_{RP,RSR,FQ,RSched}(N_0))$).

The first four models are based on simulation and therefore, their application to predicting lengths-of-stay is straightforward. Specifically, all test-day patients are placed into the simulator at their corresponding actual arrival time, and their simulated departure times are recorded. The predictor that we use per patient is the average length-of-

[5] See QueueingCPN project: https://github.com/AndreasRoggeSolti/QueueingCPN

	N_0	RP	RP&RSR	RSched	Queueing
Mean Error	0.92	0.91	0.88	0.46	-0.42
RMSE	1.97	1.95	1.95	1.82	1.38

Table 2: Mean Error and RMSE (hours) for length-of-stay predictors.

stay of that patient across 30 runs. On the other hand, the *queueing predictor* does not require simulation, and can be calculated directly as the patient arrives. The justification for using the former quantity is based on the heavy-traffic snapshot principle for networks, a well-known result from Queueing Theory [16]. The predictor was found to be empirically accurate in several recent works on queue mining [17]. The second predictor is a first-order approximation that is based on average durations and stationarity assumptions, in the spirit of the queue-length predictor in [6, 18], extended from single-station queues to networks.

Let us examine the queueing predictor in further detail. Let $\langle q_1, ..., q_k \rangle$ be the scheduled path in terms of queueing stations for the patient whose length-of-stay we wish to predict. Denote $S_i, i = 1, ..., k$ the sojourn time (delay and activity duration) of the last patient that went through station q_i (every S_i can be calculated from histories of different patients). Let L_i be the number of patients that currently occupy the ith station (queue and service), upon the patient's arrival, and let μ_i be the service rate of the ith station. The queueing predictor LOS_q can be written as follows:

$$LOS_q = \begin{cases} \sum_{i=1}^{k} S_i & \text{if } S_i > 0, \forall i \\ \sum_{i=1}^{k} \frac{(L_i+2)}{\mu_i} & \text{otherwise.} \end{cases} \quad (2)$$

As default, we use the well-established snapshot predictor, which uses the sum of recent visits to stations $q_1, ..., q_k$. However, S_i might not exist ($S_i = 0$) for some of the patients, since there is a positive probability that no other patient has visited station q_i before the arrival time of the current patient. For these cases, we resort to the second predictor, which assumes that the queue-length will not change while the patient is in the system. The second predictor assumes that for each station, the patient will wait for the queue to clear up ($L_i + 1$ service terminations), at rate μ_i. Then, the patient enters service and gets served at rate μ_i, hence the total time per station is $\frac{(L_i+2)}{\mu_i}$.

Results. Table 2 presents the results of the empirical evaluation, with time units being hours. The considered measures are Mean Error representing the *bias* of the model, and the Root Mean Squared Error (RMSE) as an indicator for model *accuracy*. We observe that the most accurate predictor in terms of RMSE is the queueing predictor. However, it is characterized with systemic under-estimation of the length-of-stay. The first 3 simulation-based models are less accurate and comparable among each other in terms of their RMSE. These predictors present an over-estimation of the length-of-stay. In contrast, the RSched folding demonstrates improvement in both RMSE and mean error, with respect to other QCSPN models.

Discussion. The empirical evaluation demonstrates that, in terms of RMSE, the efficient queueing predictor is most accurate, when compared to the simulation models. The weakness of the projected model however, is that it cannot be applied to answer

performance questions, at a granular level. For instance, consider the estimation of individual resource utilization, without relaxing the shared-resources and parallelism assumptions. Classical Queueing Networks are not expressive enough to analyze such questions, without the help of simulation. We also observe that folding of parallelism and shared resources did not have an influence on the simulation model. This can be explained by the fact that in our dataset comprises few parallel tasks, and that tasks are executed by single resources.

The mean error measure provides us with additional insights. Since the queueing predictor builds upon the RSched folding, it neglects scheduling delays and thus has a negative mean error. This causes it to under-estimate the length-of-stay. However, this relaxation may also be the reason for its superior accuracy. The latter hypothesis is supported by the fact that the error has decreased due to the removal of scheduling transitions in the RSched model. One may then conclude that, for the process in the Dana-Farber Cancer Institute, scheduling constraints are not strictly binding in the process.

Finally, after an exploratory data analysis, we found that deviations in the order of tasks are not rate. This phenomena explains the inaccuracy of the simulation models, since they assume that the sequence of tasks is not violated. However, the queueing predictors consider only the set of tasks regardless of their execution order, which explains their accuracy.

7 Related Work

We categorize related research to three classes, namely modeling formalisms, abstraction methods and process mining techniques for performance analysis.

Formalisms. Several modeling formalisms were proposed to extend Petri Net models, such that stochastic elements and queues are included [19]. For example, Queueing Petri Nets (QPN) were developed to accommodate subprocesses that encompass queueing stations [20]. However, their work does not clearly define the allowed structure for the embedded queueing network. This can result in an arbitrary large and complex Queueing Networks within the Petri Net. Our QCSPN formalism is also related to Interval Timed Colored Petri Nets [21]. In this work, we extend this formalism with stochastic durations and scheduling transitions.

Abstraction. Abstraction techniques, such as aggregation at the net level, were applied to conceal insignificant model details with respect to some analysis [23]. Furthermore, simplifying reduction rules that preserve certain properties of the original system were applied to Petri Nets. For instance, Juan et al. considered reduction rules for delay time Petri nets [24], such that timing and deadlock properties of the model were unchanged. The idea of aggregation is also encountered in the performance analysis of Stochastic Well-Formed Colored Nets [25]. The idea is to construct the symbolic reachability graph and apply an aggregation method to condense the state space for efficient analysis. These techniques are only applicable with exponential delay distributions. Our methods allow for transitions with arbitrary firing distributions, while preserving queueing related properties with scheduling transitions.

Operational Process Mining. As previously mentioned, our work relates to discovery of Petri Net models from execution logs. Rozinat et al. [4] extracted Colored Petri Net models from data by mining control-flow, case, decision and time perspectives. Rogge-Solti et al. [5] extended this framework by considering the Generalized Stochastic Petri Net formalism, with non-Markovian durations of timed transitions. However these two works did not consider the queueing perspective, but rather modeled resource induced delays as stochastic components. On the other side of the abstraction scale, research on queue mining focused on resources, without considering the process perspective [6]. In this paper, we combine the best of both worlds by integrating the queueing perspective with other process mining perspectives. Furthermore, we generalize the approach for discovering scheduled processes presented in [2]. In their work, only a single type of Queueing Networks (Fork/Join network) was considered, while our approach allows for the discovery (through projection) of an arbitrary Queueing Network.

8 Conclusion

In this paper, we address the problem of data-driven performance analysis for scheduled processes. To this end, we develop an approach that combines techniques from Queueing Theory with Colored Petri Nets and define the corresponding class of Queue-Enabling Colored Stochastic Petri Nets (QCSPN). For computational efficiency, we define folding operations that allow us to project the originating QCSPN model into the Queueing Networks formalism. Our approach was implemented and evaluated using real-world data from an outpatient cancer hospital showing the impact of model abstraction on accuracy in terms of root mean-squared error.

We consider the current work as a first step in bringing together process mining techniques that often present a high computational cost (curse of dimensionality), and efficient Queueing Theory-based techniques that ignore elements of the process perspective (curse of simplicity). In future work, we aim to extend our approach towards conformance checking of schedules via discovery of QCSPN models. Understanding where and why patients and resources deviate from schedules is of utmost importance to hospitals and other businesses, and can have an impact on performance analysis. Furthermore, we are interested in developing techniques for predicting case paths, as well as real-time prediction methods as cases progress along these paths.

Acknowledgment. We are grateful to the SEELab members, Dr. Valery Trofimov, Igor Gavako and Ella Nadjharov, for their help with data analysis. We also thank Kristen Camuso, from Dana-Faber Cancer Institute for the insightful data discussions.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Discovery and validation of queueing networks in scheduled processes. To appear in Advanced Information Systems Engineering. (2015)

3. Pinedo, M.: Planning and scheduling in manufacturing and services. Springer (2005)
4. Rozinat, A., Mans, R., Song, M., van der Aalst, W.M.: Discovering simulation models. Information Systems **34**(3) (2009) 305–327
5. Rogge-Solti, A., van der Aalst, W.M.P., Weske, M.: Discovering stochastic Petri nets with arbitrary delay distributions from event logs. In: BPM Workshops. Volume 171 of LNBIP. Springer (2014) 15–27
6. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining – predicting delays in service processes. In: Advanced Information Systems Engineering. Volume 8484 of LNCS. Springer (2014) 42–57
7. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. John Wiley & Sons (2006)
8. Vernon, M., Zahorjan, J., Lazowska, E.D.: A comparison of performance Petri nets and queueing network models. University of Wisconsin-Madison, Computer Sciences Department (1986)
9. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use. Volume 1. Springer (1997)
10. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. **2**(2) (May 1984) 93–122
11. van der Aalst, W.M.P.: Petri net based scheduling. Operations-Research-Spektrum **18**(4) (1996) 219–229
12. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11) (1983) 832–843
13. Boxma, O., Koole, G., Liu, Z.: Queueing-theoretic solution methods for models of parallel and distributed systems. Centrum voor Wiskunde en Informatica, Department of Operations Research, Statistics, and System Theory (1994)
14. Jacobson, P.A., Lazowska, E.D.: Analyzing queueing networks with simultaneous resource possession. Commun. ACM **25**(2) (February 1982) 142–151
15. Pommereau, F.: Quickly prototyping Petri nets tools with SNAKES. In: Proceedings of PNTAP'08, ACM (2008) 1–10
16. Reiman, M.I., Simon, B.: A network of priority queues in heavy traffic: One bottleneck station. Queueing Systems **6**(1) (1990) 33–57
17. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Technical report, (2014)
18. Ibrahim, R., Whitt, W.: Real-time delay estimation based on delay history. Manufacturing and Service Operations Management **11**(3) (2009) 397–415
19. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets. Springer (1996)
20. Bause, F.: Queueing Petri nets-a formalism for the combined qualitative and quantitative analysis of systems. In: PNPM'93, IEEE (1993) 14–23
21. van der Aalst, W.M.P.: Interval timed coloured Petri nets and their analysis. In: Application and Theory of Petri Nets 1993. Volume 691 of LNCS. Springer (1993) 453–472
22. Han, Y., Luo, X.: Composition and reduction of web service based on dynamic timed colored petri nets. In: Parallel and Distributed Processing with Applications, IEEE International Symposium on, IEEE (2009) 659–663
23. Smirnov, S., Reijers, H., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases **30**(1) (2012) 63–99
24. Juan, E.Y., Tsai, J.J., Murata, T., Zhou, Y.: Reduction methods for real-time systems using delay time petri nets. IEEE Transactions on Software Engineering **27**(5) (2001) 422–448
25. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans. Comput. **42**(11) (1993) 1343–1360