Discovering Queues from Event Logs with Varying Levels of Information

Arik Senderovich[2], Sander J.J. Leemans[1], Shahar Harel[2], Avigdor Gal[2], Avishai Mandelbaum[2], and Wil M.P. van der Aalst[1]

[1]Eindhoven University of Technology, the Netherlands; [2]Technion, Haifa, Israel

Abstract. Detecting and measuring resource queues is central to business process optimization. Queue mining techniques allow for the identification of bottlenecks and other process inefficiencies, based on event data. This work focuses on the discovery of resource queues. In particular, we investigate the impact of available information in an event log on the ability to accurately discover queue lengths, i.e. the number of cases waiting for an activity. Full queueing information, i.e. timestamps of enqueueing and exiting the queue, makes queue discovery trivial. However, often we see only the completions of activities. Therefore, we focus our analysis on logs with partial information, such as missing enqueueing times or missing both enqueueing and service start times. The proposed discovery algorithms handle concurrency and make use of statistical methods for discovering queues under this uncertainty. We evaluate the techniques using real-life event logs. A thorough analysis of the empirical results provides insights into the influence of information levels in the log on the accuracy of the measurements.

1 Introduction

Detecting and measuring resource queues is central to business processes. Resource queues reveal bottlenecks and violations of service-level agreements. Moreover, sojourn times and delays are basic Key Performance Indicators (KPI) that cannot be accurately calculated without queueing measurements. Process mining is a research field which aims to extract such performance information from event logs [1]. Queue mining can be seen as a particular class of process mining techniques focusing on the use of queueing theory to characterize queues from data and facilitate performance analytics [2]. Existing queue mining approaches consider event logs in which full queueing-related information is readily available. Such information includes timestamps of enqueue and service start events. However, for real-life processes, queueing information is often unavailable in the event log. Thus, previous work on the analysis of resource behavior discovered queueing information from logs with missing start timestamps, with the use of extensive resource information [3]. Moreover, both [2, 3] did not consider complex process models with parallelism, but rather worked at the perfectly-ordered log level. And in [4], missing timestamps were imputed by using a class of stochastic Petri nets. However, they consider a single timestamp per activity, without differentiating activity times from queues.

In this paper, we explore the influence of available information in the log on the accuracy of the aforementioned queue mining techniques. The event logs that we consider are general and accommodate a life cycle of three stages per activity: (1) in transition, (2) in queue, and (3) in service. To measure queues, even in case of missing timestamps for enqueueing/service start information, we assume that a process model containing the control flow of the process is given. Such a model can be discovered from the event log using a process discovery algorithm, e.g. in the form of a process tree [5] and subsequently extend the control-flow perspective with queueing measurements. The discovered model enables the extraction of a queue log, which (ideally) contains activity instances with the corresponding enqueue, start, and end timestamps.

Our methodology for measuring queue lengths, i.e. the number of work items in queue at a given point in time, starts with a queue log. First, we cluster the log according to durations (e.g. sojourn time), and get clusters with similar durations, and therefore loads. Then, we use these loads to fit a phase type distribution to life cycle durations with the use of Bayesian inference. For the evaluation of our proposed approach we applied our techniques to three real-life event logs: two call center logs and a hospital log. The results show that the queue length can be estimated using only partial information from the event log. Furthermore, the accuracy of our techniques depends on the amount of available information, and on the parameters of our techniques.

The remainder of this paper is organized as follows. Section 2 defines the event log, the role of process models in queue discovery, and the levels of available information that we assume on our event logs. Moreover, the section states the main problem that we solve in this paper. Section 3 presents our methodology for queue length discovery. Section 4 presents our experiments and discusses their results. Section 5 concludes the paper.

2 Information Levels in Event Logs & Process Models

In this section we define event logs and an activity life-cycle model. We then offer a classification of the levels of information that we consider in the event logs, and state the main problem that we solve in this paper.

2.1 Event Logs

An *event log* L is a multi-set of traces, where a *trace* represents a single case, *e.g.*, a customer or a patient visiting a hospital. A trace consists of several *activity instances*, which represent pieces of work performed for the trace. Consider the process model presented in Figure 1. A trace for a patient in this process may include blood draw and doctor examination:

$$\langle \ldots \text{blood draw}^{11:27}_{start}, \text{blood draw}^{11:29}_{complete}, \text{exam}^{11:45}_{enqueue}, \text{exam}^{12:02}_{start}, \text{exam}^{12:15}_{complete} \cdots \rangle.$$

Each of the above *events* contains a timestamp at which the event occurred, and the life-cycle transition, which denotes a change of state for the activity instance.

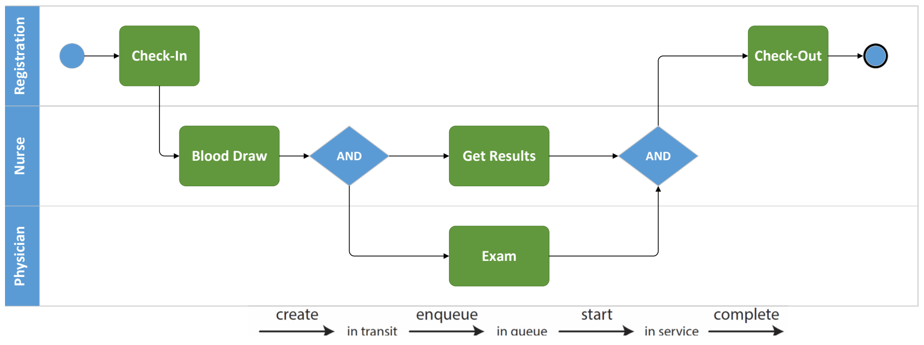

Fig. 1. An examination process in an outpatient hospital; activity life-cycle

The life-cycle transitions of the last three events, belonging to the activity instance 'exam' is as follows: the patient reports at the waiting room and enters a queue (*enqueue*); after a while she is invited into the examination room (*start*); and finally her examination finishes (*complete*). At the bottom of Figure 1 the activity life-cycle is illustrated. We assume that the transition *initiate* is not available explicitly in the event log, but that it coincides with the last completed activity instance. In our example, at 11:29, after blood draw completion, the examination was initiated and the patient started transit (she walked towards the examination room). In this paper, we aim to estimate performance measures if not all timestamps are present.

2.2 Process Models

To infer the initiate life-cycle transition, we assume it coincides with the last completed activity instance. However, this is not necessarily the activity instance of which the completion time is the closest. For instance, in our example, a lab test is processed for results in parallel with the examination activity. This would obviously have no influence on the transit time of the patient, as she has to walk from the laboratory to the examination room. Even though the lab results are ready before she reaches the room, it does not imply that she started walking later. Therefore, knowledge of the control-flow of the process is required, i.e. the initiation time of an activity is the *last completed non-parallel activity instance*. In our experiments, we used process trees [6] because there are powerful discovery techniques available for this class of models, and the models are guaranteed to be deadlock and livelock free. Process trees provide information whether activities are parallel by their structure. Other formalisms such as Petri nets and BPMN might provide this information as well.

Our approach assumes the presence of a process model to infer initiation events, to identify activity instances and to deal with deviations [7]. We do not assume that the event log describes activity instances. Therefore, in Section 3.1, we introduce a method to obtain activity instances from an event log and a process model, and show how we handle with process deviations.

2.3 Levels of Information

Not all event logs encountered in practice contain timestamps for all life-cycle transitions. Throughout the paper, we investigate the ability to measure queues lengths, as a function of the available information. More formally, let $\mathcal{I} = \{e, s, c\}$ be the event types that an event log may contain with e, s, c corresponding to enqueue, start and complete events, respectively. The level of information that is available in a given event log can be described by $I \subseteq \mathcal{I}$. We consider the following three levels of information:

– {c} Activities and completion timestamps; *e.g.*, $\langle a^{11:38}_{complete} \rangle$.
– {s,c} Activities, start and completion timestamps; *e.g.*, $\langle a^{11:35}_{start}, a^{11:38}_{complete} \rangle$.
– {e,s,c} Activities, enqueue, start, and completion timestamps; *e.g.*, $\langle a^{11:30}_{enqueue}, a^{11:35}_{start}, a^{11:38}_{complete} \rangle$.

For the latter level of information ($\{e, s, c\}$), queues in front of activities can simply be computed when a suitable process model is at hand. In the remainder of the section we state the main problem that we solve in the paper.

2.4 Problem Statement

In business processes, cases queue waiting for resources rather than for activities. However, in this work, we assume not to have enough data to describe the resource perspective completely. Therefore, we aim at measuring queues in front of activities, which is, in a sense, a more informative view than the one that would be provided by the resource perspective, since we also incorporate the notion of control-flow.

Let M be a process model with $\mathcal{A}_M$ being the set of activities present in the model and the log. Denote L_I the event log with information level I, and let $[0, T]$ be the time interval such that 0 is set to be the time of the first measured activity in the log and T being the time of the last measured activity (we use a continuous notation for simplicity). For simplicity we shall assume that the exogenous arrival time of each case is known. The *queue measurement - activities* (QMA) problem receives a model M, and a log, L_I. The solution to the problem is a quantifier for queue lengths (exact or approximated) for each activity in the model at any point in time, namely $\tilde{Q}_A(t),\ A \in \mathcal{A}_M,\ t \in [0, T]$. For example, in [2], this problem is solved by trivially deriving the queue length from a full information log ($\{e, s, c\}$).

3 Methodology and Algorithms

This section describes our methodology and algorithms for solving the QMA problem. The approach is shown in Figure 2: the first step in the methodology

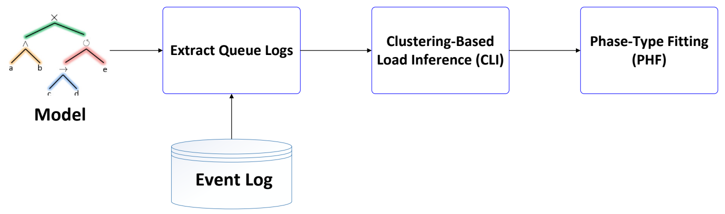

Fig. 2. A methodology for queue measurement in logs with partial information

is to extract a queue log per activity, containing the execution information for that activity. The queue log then serves as an input to a clustering step, which in turn is used both for load inference, and for a first-attempt measurement of the queue length. The result of the clustering algorithm is the input for our statistical method, namely, the phase type fitting method, which fits a stochastic model to the event log via Bayesian inference. Below, we present the technique for extraction of a queue log, define queueing quantification, and present our techniques for measuring queue lengths.

3.1 Extracting Queue Logs

The first step is to split the event log, L_I, into several activity queue logs, L_A, one for each activity, $A \in \mathcal{A}_M$. An activity queue log describes the execution information for all activity instances of that activity, namely for each execution of A in the event log, a queue log contains all activity instances of A that were recorded in the event log [2]. For simplicity, we assume that each activity is performed by a single resource type, and the resource type performs a single activity. Therefore, we may consider a queue log per each activity separately.

Obtaining queue logs requires a few steps. First, deviations between the event log, L_I, and model, M, should be removed. Then, activity instances should be derived from the events in L_I. This step yields the timestamps of *enqueue*, *start* and *complete*. Subsequently, activity instances are be mapped to activities, according to M. Last, the *initiate* timestamps should be derived from the activity instances and M. In this step, knowledge of concurrent activities is necessary. We perform the first three steps by computing an optimal alignment, cf. [7]. The last step is performed by inspection of M, and detection of parallel activities. Note that the relation between traces and activity instances does not exist in the resulting queue log, and therefore we regard each activity instance as an independent queueing entity.

3.2 Notation and Queue Quantification

In this part, we present notation that we use when describing our methods and algorithms. Then, we provide a quantification framework for $\tilde{Q}_A(t)$, which is the log-based solution to the QMA problem.

Let $\mathcal{T}_j^A$ be a random function that maps activity instances from log L_A to timestamps of the jth element of the life-cycle denoted $j \in \{init, enqu, start, comp\}$. When a timestamp for element j is available, the random functions becomes deterministic, denoted $\tau_j^A : L_A \to \mathbb{TS}$. Note that for the considered information levels, τ_{comp} is assumed to be know, while τ_{init} is obtained during the extraction of the queue log.

We denote $X_n^A(t), t \in [0, T]$ the stochastic process that corresponds to the cumulative time that the nth activity instance spends in A at time t, namely $X_n^A(t) = t - \mathcal{T}_{init}^A(n)$. The realization of $X_n^A(t)$, denoted $x_n^A(t)$, can be obtained by the value $t - \tau_{init}^A(n)$, i.e. the time spent between initiation and t.

An activity instance n is in queue for A at time t whenever the following probabilistic event holds: $q_n^A(t) = \{\mathcal{T}_{enqu}^A(n) \leq t \leq \mathcal{T}_{start}^A(n)\}$. Denote $\mathbb{1}_{q_n^A(t)} | X_n^A(t) > x_n^A(t)$, the random variable that indicates whether activity instance n is enqueued in A at time t, conditioned on a cumulative length of stay of $x_n^A(t)$. Clearly, when the log contains the enqueue timestamp the indicator is constant (either zero or one), since the values of $\tau_{enqu}^A(n), \tau_{start}^A(n)$ are known.

For the general case, of various levels of information, we define the following quantity to measure queue-lengths:

$$\tilde{Q}_A(t) = \sum_{n \in L_A} \mathbb{E}[\mathbb{1}_{q_n^A(t)} | X_n^A(t) > x_n^A(t)], \tag{1}$$

for every activity and time in the log, with $\mathbb{E}$ being the expectation of a random variable. In other words, we quantify the queue length in-front of activity as the sum of expected values of the indicator. The quantifier for queue length, $\tilde{Q}_A(t)$, can be written as follows:

$$\mathbb{E}[\mathbb{1}_{q_n^A(t)} | X_n^A(t) > x_n^A(t)] = P(q_n^A(t) | X_n^A(t) > x_n^A(t)). \tag{2}$$

For each $n \in L_A$, the right-hand side part of Equation (2) is quantified in the techniques that follow.

3.3 Clustering-Based Load Inference

The idea behind the clustering step is to categorize each trace of the activity queue log, L_A, according to the observed total durations, i.e. $\tau_{comp}^A(n) - \tau_{init}^A(n)$, $\forall n \in L_A$. For the completes only information level ($\{c\}$) total durations are sojourn times, while for the $\{s, c\}$ the total durations are time in transit and in queue. The main assumption of this method is that one of the resulting clusters contains traces of L_A for which queue size in-front of activity A was 0. In other words, we assume that the resources are not working 100% of the time for the entire time period recorded in the queue log. All other clusters are expected to contain traces that were enqueued during their stay.

We are now ready to present our Clustering-based Load Inference (CLI) algorithm, with clustering being performed on the feature: observed total durations. Denote $1, ..., K$ the cluster indexes with K being a predefined number of clusters.

As a first step, a K-Means algorithm runs to partition the traces in the log, L_A, into sub-logs L_A^k (see [8] for references on clustering techniques). Following the aforementioned assumption, L_A^1 contains only non-queueing cases. Other clusters, $k = 2, ..., K$, are assumed to contain traces that were enqueued, with time in queue increasing as k approaches K. The clusters represent K variants of system load, which we then pass on to the next two methods of Figure 2. Note that the selection of K is process dependent, e.g. in call centers, the load is often partitioned into 4 clusters: no load, typical load, moderate load, heavy load [2].

The result of clustering can already be used to construct a first-attempt quantifier for the queue length. Specifically, we obtain the probability for an activity instance to be in queue by applying Bayes' Theorem to Equation (2):

$$P(q_n^A(t) \mid X_n^A(t) \ > \ x_n^A(t)) = \frac{(1 - \pi(n \in L_A^1)) \ P(X_n^A(t) \ > \ x_n^A(t) \mid q_n^A(t))}{P(X_n^A(t) \ > \ x_n^A(t))}, \tag{3}$$

with $\pi(n \in L_A^1)$ being the prior probability of a case belonging to the non-queueing cluster, $P(X_n^A(t) > x_n^A(t) \mid q_n^A(t))$ being the probability for the current time in activity being longer than $x_n^A(t)$ given that an activity instance, n, is in queue, and $P(X_n^A(t) > x_n^A(t))$ being the probability for total time being longer than $x_n^A(t))$. The three components of Equation (3) can be easily estimated by using the corresponding Maximum Likelihood Estimates (MLE). The results serve an input to the next phase of our methodology, namely the phase type fitting algorithm.

3.4 Phase Type Fitting

In this part, we assume that each step of the activity life-cycle has an exponential duration, with a rate that changes per each cluster (depends on system load). Moreover, we assume that the times a case spends in each step are independent. The assumption that the time in transit and in service is exponential is quite common in queueing literature [10, 11]. However, waiting times in queue were shown to be exponential only for several specific queueing models; e.g., the time in queue for $M/M/n$ queues, conditioned that a case waits, is exponential. Another example in which waiting times are exponentially distributed is for queues in heavy-traffic [12]. Nevertheless, since we allow the rate of the exponential distribution per each component of the life-cycle to vary among clusters, we assume, as an approximation, that the time in queue is indeed exponential.

Under these assumptions, the suitable model for our life-cycle is that of a continuous-time Markov-chain (CTMC) that has $S = 3$ states in sequence. The chain absorbs after going through the three stages (transit, queue, service) when the activity terminates. The total time that it takes for a case to go through the life-cycle (i.e. the sojourn time) has a phase type distribution [9]. The phase type distribution is characterized by two parameters: (1) a vector of initial states, η, of size S (e.g. for the $\{c\}$ level of information all traces start in the first phase, and thus $\eta = (1, 0, 0)$), and (2) a transition-rate matrix $\boldsymbol{G}$ that describes the rates with which the CTMC moves between states.

Several techniques were suggested to fit a phase type distribution to data, when the available measurements include total time in process [13]. In our work, we use a Bayesian approach that relies on a Markov-Chain Monte-Carlo method, which we refer to as K-Phase Type Fitting (K-PHF) [14]. The K in the algorithm comes from the number of clusters that it receives from CLI. The algorithm is based on the structure of the phase type process, the initial state η, and prior distributions for the transition rates. The output is the matrix $\widehat{\boldsymbol{G}}$, estimated based on the data. In this work, we used priors that assume that the total time is uniformly divided among the stages of the life-cycle. For example, consider the completes only information level ($\{c\}$), a case spends 1/3 of the total time in each of the stages: transit, queue, and service. Thus, 1/3 serves as the prior to our K-PHF method.

The output of the K-PHF algorithm is used to estimate the probability from Equation 2, for each trace individually. It is well-known that the probability for the phase type process to be in state s with $i = 1, ..., S$, given that the elapsed time in process is x is given by $\eta \exp(x\boldsymbol{G})_s$. The expression $\exp(x\boldsymbol{G})_s$ is the s column of the matrix exponential for the original transition-rate matrix, $\boldsymbol{G}$, multiplied by x. Thus, for every point in time $t \in [0, T]$:

$$P(q_n^A(t) \mid X_n^A(t) \; > \; x_n^A(t)) = \eta \; \exp(x_n^A(t)\boldsymbol{G})_s, \tag{4}$$

with the right-hand side expression easily obtained from the K-PHF algorithm. We fit a phase type distribution to each of the clusters, namely $k = 1, ..., K$, and consequently get a corresponding transition matrix $\boldsymbol{G}^k$ per cluster. Last, we use the latter to calculate the expression in Equation (4).

4 Evaluation

Finally, in order to evaluate our approach and illustrate the influence of different log information levels, we perform experiments using three real-life event logs with full information ($\{e, s, c\}$). We removed events from these logs (resulting in $\{s, c\}$ and $\{c\}$) to test how close the techniques described in this paper come to the actual queue length. The first two logs (L_1 and L_2) originate from a call center of an Israeli bank, while the third log (L_3) comes from an outpatient hospital.

4.1 Datasets

Logs L_1 and L_2 both contain a process in which customers call the bank and are initially answered by a computer system, namely the Voice Response Unit (VRU), which is designed to handle basic questions and requests. If the customer requires further service beyond the VRU, she can choose to join the queue. Customers can abandon at any moment while they are in queue, and can also leave the VRU without reaching the queue. However, we filter these customers out of the original log, since they do not match our assumption on activity life-cycle Section 2. None of our techniques directly uses the dependency between the customers in queue, and therefore this filtering does not reduce the validity of our experiments. The queue that we measure in this paper is the queue in-front of the human agents. The entire time in VRU is considered transit. Log L_2 is derived from L_1 by setting the transit time to 0.

To highlight the parallelism-handling capabilities of our approach, log L_3 originates from a process that contains parallelism: patients being examined and undergo several concurrent procedures. Of this process, the activity 'exam' was selected for queue measurement, as it is the most constrained activity in the process.

4.2 Experimental Setup

We start with one of the event logs, L_1, L_2 or L_3, and derive the corresponding queue log L_A of each activity A by using the method described in Section 3.1. Then, a model is discovered by Inductive Miner - infrequent [15] using default settings. Subsequently, we apply CLI and K-PHF to L_A in order to quantify $\tilde{Q}_A(t)$ (using a milisecond resolution). Similarly, the real queue length, $Q_A(t)$, is computed, using the enqueue and start timestamps. To evaluate the accuracy of the techniques, we consider the root mean squared error and the bias. First, the root mean squared error (RMSE) is computed per activity:

$$\sqrt{\frac{1}{T} \cdot \sum_{t \in [0,T]} (\tilde{Q}_A(t) - Q_A(t))^2}.$$

As a second measure, we consider the bias, which can indicate systemic error in our techniques:

$$\frac{1}{T} \cdot \sum_{t \in [0,T]} (\tilde{Q}_A(t) - Q_A(t)).$$

This procedure is applied to all methods that we described in Section 3, all levels of information, and the three event logs. In addition, we provide two baselines for comparison. First, we added the Busy Periods method (BP), which is based on [10, 3]. For time $t \in [0, T]$, BP simply counts the number of cases that are currently in the activity. BP assumes the following: (1) full knowledge of busy periods, namely times in which all resources are busy, (2) the number of resources are constant and known, and (3) in busy periods, resources are work-conserving (non-idling). We provide the algorithm with an approximation to busy periods per activity $A \in \mathcal{A}_M$, since we consider times in which $Q_A(t) > 0$ (a busy period can start with an empty queue, and all resources busy).

As a second baseline, we considered the Uniform Prior method (UP), which divides the number of cases in the activity at timestamp t by the number of unknown time intervals. For example, if the log contains the initiate and completion timestamps, UP divides the total time spent in the activity by 3, similarly to the prior that we consider for K-PHF. In case that all information is known ($\{e, s, c\}$), UP counts the precise number of cases in queue.

Table 1. Queue length evaluation.

		BP		UP		1-PHF		CLI		4-PHF	
		RMSE	bias	RMSE	bias	RMSE	bias	RMSE	bias	RMSE	bias
L_1	{c}	21.18	11.91	8.11	-5.12	10.68	-7.76	25.96	-21.34	6.20	-1.63
	{s, c}	"	"	4.63	0.83	8.08	4.04	4.22	-3.17	8.45	4.81
L_2	{c}	13.05	6.18	5.99	-2.40	13.85	8.97	20.98	-16.99	6.53	1.56
	{s, c}	"	"	7.34	4.87	9.23	5.77	2.63	2.00	11.93	7.96
L_3	{c}	21.38	-17.51	4.08	1.20	6.84	5.32	10.69	7.25	8.58	6.98
	{s, c}	"	"	4.86	3.22	10.37	8.62	10.80	8.31	12.48	10.47

For our clustering-based methods that we present in Section 3, i.e. CLI and K-PHF, we have selected K to be 4, which is a standard selection for call center data [2]. For K-PHF we also consider $K = 1$, to test the impact of load knowledge on the goodness-of-fit for the phase type distribution. To investigate the influence of K on the clustering-based methods, the procedure was repeated for CLI with various values of K.

4.3 Results and Discussion

Table 1 shows the results for different combinations of event logs, information levels and techniques. Figure 3 shows the influence of K on CLI, tested on L_1. Last, Figure 4 shows the queue length obtained by 1-PHF, and 4-PHF (best algorithm for complete only information) as function of time, compared to the real queue length.

Considering log L_2, our experiments show the sensitivity of BP in case assumptions (2) and (3) do not hold: for L_2, with no transit time and known busy periods, BP overestimates the log considerably (L_2 RMSE 13.05; bias 6.18). This is hardly surprising when considering resource dynamics in real-life settings: resources go on breaks, and start/end their shifts during the day. Surprisingly, UP is superior in all $\{c\}$ scenarios, except for 4-PHF.

We notice that CLI performs poorly for the $\{c\}$ level of information. A sensitivity analysis revealed that indeed $K = 4$ is not optimal for CLI with completes only, and it can be improved by selecting $K = 2$ (Figure 3). However, for the $\{s, c\}$ scenarios and the two call center logs (L_1, L_2), the results are superior to all methods.

Across our experiments, 1-PHF performs mediocre on performance, since it neglects differences in system load. When changes in the load are considered, we indeed see an improvement, and 4-PHF is indeed the superior algorithm across the $\{c\}$ information level. Intuitively, one would expect that methods perform better when given more information. However, 4-PHF performs best when given only completion timestamps; when given more information, i.e. start events, 4-PHF performs worse. We suspect that our choice of K in CLI might be of influence here.

Inspecting only averaged results can be misleading. Thus, we turn our attention to Figure 4, and observe the behavior of queue lengths of two methods:

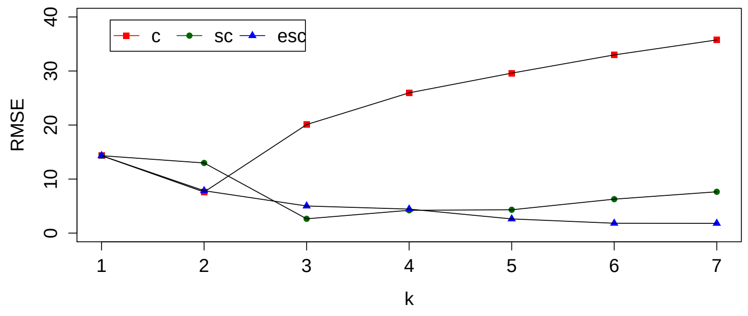

Fig. 3. Influence of K on CLI with L_1.

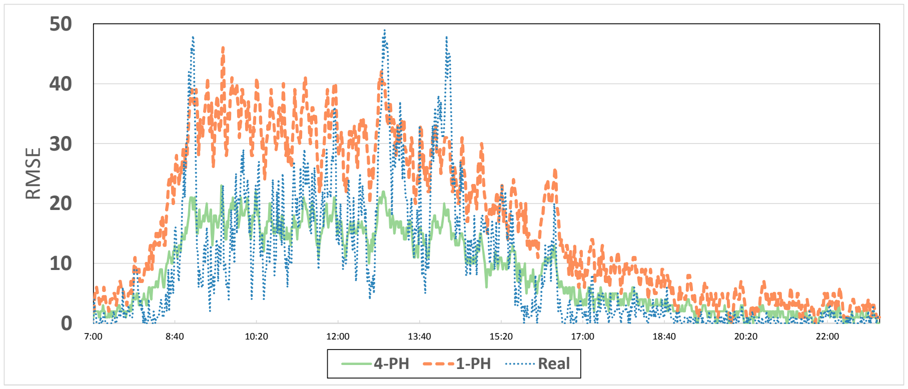

Fig. 4. Queue length for 1-PHF, 4-PHF and real with L_1; the x-axis represents time (one day).

1-PHF and 4-PHF, under knowledge of completion timestamps. We observe that 1-PHF is able to capture the first peak of the day. However, it misses the remainder of the day by overestimating the queue length, especially for the peak period. In contrast, 4-PHF captures much of the behavior, except for sudden changes in queue lengths.

5 Conclusion

In this paper, we showed how to discover queue lengths for activities in operational processes. Specifically, queue lengths were quantified for event logs with varying levels of information. In particular, we proposed a comprehensive approach, which includes the use of a process model and recorded event data to derive a queue log. The queue log then feeds a sequence of two techniques for measuring the queue length. The first technique is based on K-Means clustering, while the second technique is based on phase type fitting. We performed a thorough evaluation of our methods against baselines and presented the deviations from the real measurements. We tested the methodology on three real-life logs: two call centers logs, and one log of an outpatient hospital. The results show that our methods are able to discover queue lengths with various levels of accuracy. This accuracy is sensitive to the level of information, and to the K of the clustering algorithm.

In future work, we intend to take the resource perspective into account, since cases wait for resources and not for activities. We aim at utilizing the information on the matching between activities and the corresponding resources that is often available in event logs to improve the accuracy of our methods. Furthermore, we aim to consider the dependencies between queued cases in a more rigor way, by discovering queueing models that correspond to the resources involved.

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.: Queue mining for delay prediction in multi-class service processes. Technical report, Tech. rep (2014)
3. Nakatumba, J.: Resource-Aware Business Process Management: Analysis and Support. PhD thesis, Eindhoven University of Technology (2013)
4. Rogge-Solti, A., Mans, R., van der Aalst, W.M.P., Weske, M.: Repairing event logs using timed process models. (2013) 705–708
5. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In Colom, J.M., Desel, J., eds.: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013.
6. Buijs, J., van Dongen, B., van der Aalst, W.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation, IEEE (2012) 1–8
7. Adriansyah, A.: Aligning Observed and Modeled Behavior. PhD thesis, Eindhoven University of Technology (2014)
8. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA (2001)
9. Neuts, M.F.: Renewal processes of phase type. Naval Research Logistics Quarterly **25**(3) (1978) 445–454
10. Mandelbaum, A., Zeltyn, S.: Estimating characteristics of queueing networks using transactional data. Queueing systems **29**(1) (1998) 75–127
11. Mandelbaum, A., Zeltyn, S.: Service engineering in action: the Palm/Erlang-A queue, with applications to call centers. In: Advances in services innovations. Springer (2007) 17–45
12. Kingman, J.: On queues in heavy traffic. Journal of the Royal Statistical Society. Series B (Methodological) (1962) 383–392
13. Asmussen, S.: Phase-type distributions and related point processes: Fitting and recent advances. In: International Conference on Matrix-Analytic Methods in Stochastic Models. (1996) 137–149
14. Aslett, L.J., Wilson, S.P.: Markov chain monte carlo for inference on phasetype models. ISI (2011)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In Volume 171 of Lecture Notes in Business Information Processing., Springer (2013) 66–78