Data-Driven Process Performance Measurement and Prediction: A Process-Tree-Based Approach

Sebastiaan J. van Zelst[1,2], Luis F.R. Santos[2], Wil M.P. van der Aalst[2,1]

[1] Fraunhofer Institute for Applied Information Technology (FIT), Germany
sebastiaan.van.zelst@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany

Abstract. To achieve operational excellence, a clear understanding of the core processes of a company is vital. Process mining enables companies to achieve this by distilling historical process knowledge based on recorded historical event data. Few techniques focus on the prediction of process performance after process redesign. This paper proposes a foundational framework for a data-driven business process redesign approach, allowing the user to investigate the impact of changes in the process, w.r.t. the overall process performance. The framework supports the prediction of future performance based on anticipated activity-level performance changes and control-flow changes. We have applied our approach to several real event logs, confirming our approach's applicability.

Key words: Process mining, Process improvement, Process redesign

1 Introduction

Information systems, e.g., Enterprise Resource Planning (ERP), support the execution of a company's core processes. These systems capture at what point in time an activity was performed for an instance of the process. *Process mining* techniques turn such *event data* into actionable knowledge [1]. For example, various *process discovery techniques* exist that transform the event data into a *process model* describing the process behavior as captured in the data [2]. Similarly, *conformance checking techniques* quantify whether the process behaves as recorded in the event data w.r.t. a given reference model [3].

The overarching aim of process mining techniques is to *improve the process*, e.g., decreasing the process duration while maintaining the same quality level. Yet, a relatively small amount of work focuses on data-driven techniques to support decision-makers in effectively improving the process. For example, in [4], the authors propose to discover *simulation models* on the basis of recorded event data, which can be used to simulate the process under different "What if" scenarios. In [5], a similar approach is proposed, explicitly focusing on macro-level aspects of the process, e.g., average case duration. The work presented in this paper acts in the middle of the two spectra covered by the work mentioned. Similar to [4], we measure performance on the *activity-level*. However, we do not learn a complete simulation model. Instead, we explain the historical behavior captured

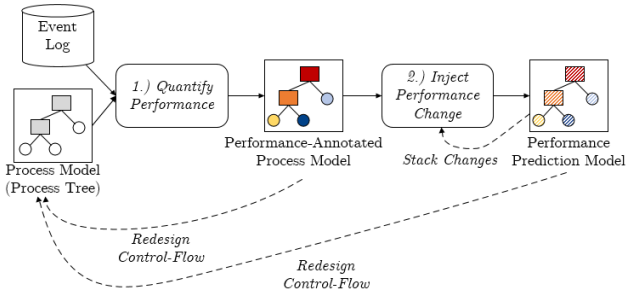


Fig. 1: Overview of our proposed framework. The current process performance is quantified in the context of a (given) process model. Anticipated performance changes are injected to compute possible future performance.

in the event log in the context of a model specifying the process behavior. We use the annotated model for the prediction of future behavior.

In Fig. 1, we depict the proposed framework. An event log and a process model act as the input artifacts. We compute *timed partial order alignments*, which we use to quantify the process's historical performance in the context of the given model. Our framework supports the assessment of changes in the time-performance of activities (either waiting or service time), and it supports stacking multiple anticipated improvements of the process. Since our framework takes an arbitrary process tree as an input, it is possible to extend it to calculate the effect of control-flow changes. We have evaluated a prototypical implementation of our framework using several collections of real event logs. Our experiments confirm that our framework allows us to identify the main bottlenecks of the process. Furthermore, we observe that, in some cases, the process model used as an input influences the measured performance of the bottlenecks identified.

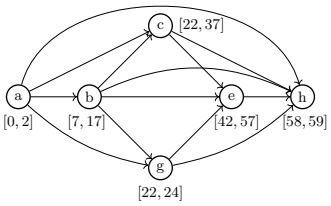
The remainder of this paper is organized as follows. Section 2 discusses related work. In Section 3, we present background notions. In Section 4, we present our framework, which we evaluate in Section 5. Section 6 concludes this paper.

2 Related Work

We refer to [1] for an overview of process mining. Most papers on prediction, focus on *intra-case prediction*, e.g., see [6]. Early work, e.g., [7], learns and uses annotated transition systems to predict possible future states of running processes. In [8], LSTM neural networks for predicting the next activity/remaining time for a process instance are studied. Data-driven global performance measurement and prediction are studied less intensively. In [9], the authors structure the field and identify the lack of relevant work in this space. Arguably the first work in this domain, i.e., [10], proposes to learn simulation models. In [11], a generic framework describing the integration of data-driven simulation models in process improvement/redesign is presented. More recently, the application of *system dynamics modeling* in the context of process mining has been studied [5].

Case-id	Instance-id	Activity name	Starting time	Finishing time	...
⋮	⋮	⋮	⋮	⋮	...
7	35	Register(a)	2020-01-02 12:23	2020-01-02 12:25	...
7	36	Analyze Defect(b)	2020-01-02 12:30	2020-01-02 12:40	...
7	37	Inform User(g)	2020-01-02 12:45	2020-01-02 12:47	...
7	38	Repair(Simple)(c)	2020-01-02 12:45	2020-01-02 13:00	...
8	39	Register(a)	2020-01-02 12:23	2020-01-02 13:15	...
7	40	Test Repair(e)	2020-01-02 13:05	2020-01-02 13:20	...
7	41	Archive Repair(h)	2020-01-02 13:21	2020-01-02 13:22	...
8	42	Analyze Defect(b)	2020-01-02 12:30	2020-01-02 13:30	...
⋮	⋮	⋮	⋮	⋮	...

(a) Example event log.



(b) POC of Case 7 of Fig. 2a.

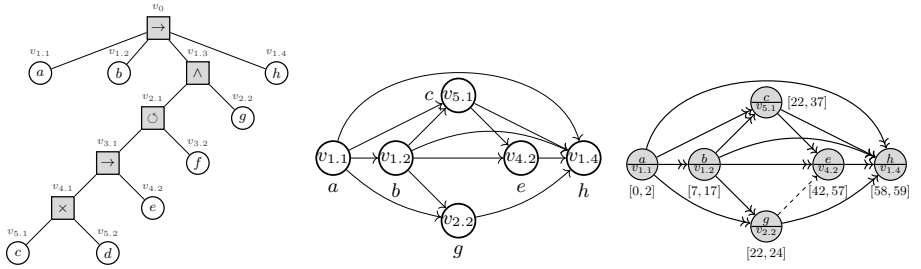
Fig. 2: Example event log (Fig. 2a) and Partially-Ordered Case (POC) (Fig. 2b).

3 Background

Event Data; Information systems store the historical execution of processes in *event logs*. In Fig. 2a, we depict an example event log. Each row refers to an *activity instance* describing an executed activity. Activity instances describe several data attributes, e.g., the activity name, timestamps, resource, etc. The first row of Fig. 2a describes an instance with id 35, describing activity *Register*, executed on January 2nd 2020, from 12:23 until 12:25, in the context of a *process instance* with identifier 7. Activity instances referring to the same process instance compose a *case*, e.g., in the context of case-id 7: *Register(a), Analyze Defect(b), Inform User(g), Repair(Simple)(c), Test Repair(e), Archive Repair(h)*. Hence, *a case describes a collection of activity instances*. Since activity instances record a start and an end time, they may overlap in time, e.g., consider instances 37 (Inform User) and instance 38 (Repair (Simple)). We assume a *strict partial ordering* (an irreflexive, anti-symmetric and transitive relation) of the activity instances that belong to a case. In Fig. 2b, we depict a *Partially Ordered Case* (POC) representation for Case 7. An *event log* is a collection of cases.

Process Trees; We use *process trees* as a process modeling formalism, i.e., rooted trees in which the internal vertices represent control-flow constructs and the leaves represent activities. In Fig. 3a, we depict an example process tree. The *sequence* operator ($\rightarrow$) specifies sequential behavior, i.e., first its left-most child is executed, then its second left-most child, etc. The *exclusive choice* operator ($\times$) specifies an exclusive choice between its children. Parallel behavior is represented by the *parallel operator* ($\wedge$), i.e., all children are executed simultaneously/in any order. Repetition is represented by the *loop operator* ($\circlearrowleft$). The $\rightarrow$, $\times$, and $\wedge$-operator can have an arbitrary number of children. The $\circlearrowleft$-operator has exactly two children. Its left child is always executed, i.e., at least once. When executing its right child, we again execute its left-most child to finish the operator. We assume that a process tree describes a set of strict partial orders as its language, e.g., in Fig. 3b we depict one Q_1. Due to the loop operator ($v_{2.1}$), the process tree in Fig. 3a describes an infinite amount of LPO's.

Partially-Ordered Alignments; Alignments [3, Chapters 7-9] quantify the behavior captured in an event log in terms of a reference process model. We consider *Partially-Ordered Alignments* (POAs) [12]. POAs align a POC with a partial or-



(a) Example process tree Q_1. Leaf vertices describe activity labels, internal vertices describe control-flow operators.

(b) Labeled Partial Order (LPO) that is in the language of Q_1.

(c) Partially-Ordered Alignment (POA) of the POC in Fig. 2b and the LPO in Fig. 3b.

Fig. 3: Example process tree (Fig. 3a) and a member of its language (Fig. 3b).

der in a process model's language. The elements of alignments are called *moves*. An observed activity for a case that is also described by the process model is referred to as a *synchronous move*, e.g., for the POC in Fig. 2b the first activity instance describes activity a, which is in line with any partial order described by Q_1. We record a *synchronization* as a tuple $(a, v_{1.1})$ aligning the observed activity instance with label a, with the execution of vertex $v_{1.1}$. If an activity occurred that is not described by the model, we write $(a, \gg)$, i.e., referred to as a *log move*. If the model describes behavior that is not observed, written as $(\gg, v)$ (here v is some leaf node), we refer to a *model move*. The ordering among the moves is extracted from both the POC and the model. In Fig. 3c, we depict a POA of the POC in Fig. 2b and the partial order in Fig. 3b, i.e., only describing synchronous moves. The double-headed arrows represent ordering relations that are both described by the process model and the POC. The single-headed dashed arrow represents an order relation that is only present in the POC.

4 POA-Based Performance Measurement and Prediction

Here, we present our framework for data-driven process performance measurement and prediction. We focus on time-based performance, i.e., *waiting*, *service*, *idle* and *cycle time*. These metrics are schematically visualized in Fig. 4. In the remainder, we describe the steps of the approach: *1.) Performance Quantification* and *2.) Performance Change Injection* (cf. Fig. 1).

4.1 Performance Quantification

To measure time performance, i.e., as recorded in an event log and conditional to a given process model, we use the notion of *Timed Partially-Ordered Alignments* (TPOA). In a TPOA, the moves of a POA are associated with timestamps

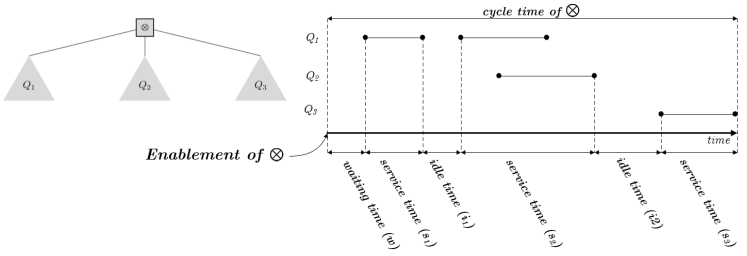


Fig. 4: Overview of the performance metrics considered.

derived from the event log. For synchronous and log moves, the time range is copied from the event log. Model moves are interpreted as *point-intervals*, i.e., having a zero-duration (this design decision is easily changed, e.g., by taking the log-based average duration of the activity described by the model move). To compute a point-interval for a model move, we obtain the maximum interval value x of any of its predecessors in the POA, i.e., according to the model's ordering-relations, which is used as a point-interval of the form $[x, x]$ or $[0, 0]$ if the move has no predecessors. In the remainder of this section, we describe and exemplify the computation of the three core metrics considered for an arbitrary subtree of a process tree and a given trace, i.e., based on timed moves.

Service Time; The service time of a (sub)tree comprises all intervals at which it is *active*, i.e., work has been performed. In Fig. 4, the service time of the root operator, i.e., $\otimes$, consists of three time ranges, i.e., s_1, s_2 and s_3. The service time range s_2 consists of the service times observed for Q_2 and Q_3. In the running example (Fig. 3c), the service time of $v_{3.1}$ comprises the service times of moves $(c, v_{5.1})$ and $(e, v_{4.2})$, i.e., ranges $\{[22, 37], [42, 57]\}$. The service time of $v_{1.3}$ is the same, i.e., $(g, v_{2.2})$ is executed concurrently with $(c, v_{5.1})$.

Waiting Time; The waiting time of a (sub)tree, i.e., w in Fig. 4, is the time between the tree's enabling until the first activity observed in any of its children. Given a subtree Q' of a process tree Q, its waiting time is computed by subtracting the *minimum* starting time of any of the model/synchronous move related to Q', from the *maximum* finishing time of any synchronous/model move preceding any move related to Q'. Consider move $(e, v_{4.2})$ in Fig. 3c. The move starts at time 42. The maximum finishing time of any move preceding the move is 37 recorded by $(c, v_{5.1})$. Hence, the waiting time of the move is captured by the range $[37, 42]$. We ignore the finishing time of move $(g, v_{2.2})$, since $v_{2.2}$ does not precede $v_{4.2}$, i.e., their common ancestor is $v_{1.3}$ (parallel operator).

Idle Time; Idle time comprises all time ranges in which we observe no activity, yet the process tree has been active before and has not finished yet. For example, the waiting time of $(e, v_{4.2})$, i.e., $[37, 42]$ represents the idle time of the subtree formed by $v_{3.1}$. We obtain the idle time of an arbitrary subtree by taking the union of all active times of the moves that are part of it, i.e., according to the POA's timed moves. Subsequently, we intersect the *complement* of the

aforementioned set of ranges with the range formed by the first start-time and maximum end-time of any of the subtree's moves.

Cycle Time; Each observed instance of a subtree Q' of some process tree Q, generates a singleton waiting time interval as well as a set of service and idle times respectively. Hence, the *cycle times* of a subtree are calculated by computing the union of the related sets of waiting, service and idle time.

4.2 Performance Change Injection

We assume that the process owner calculates the effect of changing the process's time-performance at the *activity level*, e.g., assessing the impact of a *waiting time reduction of* 20% of an activity of the process. Given an expected change (either positive or negative), we assume that all other events behave the same. Adopting such a scenario for performance prediction translates to *shifting* the time range of the timed moves, according to the desired improvement. Consider improving the performance of activity c in the running example (Fig. 2b and Fig. 3c) by 20%. This results in a new service time range of $[22, 34]$ ($0.8{\cdot}(37-22){=}12$). We shift the range of moves $(e, v_{4.2})$ and $(h, v_{1.4})$ by 3, i.e., to $[39, 54]$ and $[55, 56]$ respectively. Hence, the reduction on activity c, yields a ∼5% reduction in the overall flow-time ($\frac{3}{59}$).

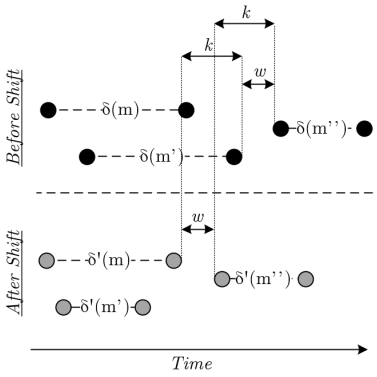


Fig. 5: Computation of the new start point of a move (m''). Before shifting, move m' directly precedes m'', after shifting, it precedes m''.

We use a *move shift function*, describing how to shift a timed move based on a proposed change. The core idea is to maintain a shift value on both the *start* and *end time* of a move of a TPOA. The shift function allows us to derive the (new) interval boundaries described by the timed move. Given some move m with time interval $[a, b]$ and a corresponding shift function value x for the start and y for the end time of m. The new time interval for m, is equal to the interval $[a+x, b+y]$ (shift forward in time: $x{<}0$ and $y{<}0$). Moves that have no predecessors in the TPOA are not shifted or shifted on their start/end time according to the performance change, e.g., a 5% reduction of service time on $(a, v_{1.1})$ in Fig. 3c, yields a shift on its end time of $2-0.95{\cdot}2{=}0.1$. For a move m that does have predecessor moves, first, a new time range for all its predecessors is computed, i.e., by applying (accumulated) shifting on top of the initially recorded time annotation of the TPOA. The initial shift values of move m are the difference between the maximum end point of its predecessors excluding any shift (i.e., based on the original time ranges of the predecessors) and the maximum ending point of its predecessors including any shift (i.e., based on the new time ranges of the predecessors). If a move relates to an activity with an anticipated performance change, the change is computed on top of the initially computed shift values. Fig. 5 shows an exemplification of the computation.

Table 1: Experimental results. Measured performance is in hours (rounded). The impact of the bottleneck reductions is relative to the original cycle time.

Event Log	*Discovery Threshold*	*Detected Bottleneck*	*Avg. Bottleneck Sojourn Time*	*Abg. Overall Cycle Time*	*Rel. Cycle Time Red. (1% Bott. Red.)*	*Rel. Cycle Time Red. (2.5% Bott. Red.)*
BPI 2017 [14]	10%	O_Cancelled	479	397.6	0.57355%	1.43387%
	60%	O_Cancelled	479	397.6	0.57355%	1.43387%
BPI 2020 Domestic Declarations [15]	10%	Declaration APP...	107	304.8	0.27794%	0.69486%
	30%	Declaration APP..	107	304.8	0.27794%	0.69486%
BPI 2020 International Declarations [15]	10%	Start trip	500.7	1244.8	0.40223%	1.00558%
	20%	Start trip	500.7	1244.8	0.39700%	0.99251%
BPI 2020 Request Payment [15]	10%	Payment Handled	102.3	315.7	0.32398%	0.80995%
	20%	Payment Handled	102.3	315.7	0.32398%	0.80995%
BPI 2020 Travel Permit [15]	10%	Send Reminder	1249.4	1331	0.19091%	0.47728%
	20%	Send Reminder	1349	1331	0.16858%	0.42145%
Road Traffic [16]	10%	Send for Credit...	11704.8	6976.8	0.66604%	1.66510%
	20%	Send for Credit...	11704.8	6976.8	0.66604%	1.66510%
Hospital Billing [17]	10%	FIN	560.2	556.7	0.65105%	1.62763%
	20%	FIN	560.2	556.7	0.65105%	1.62763%

5 Evaluation

In this section we evaluate our approach. We conducted our experiments using a publicly available implementation of our framework (https://github.com/luisfsts/KPIAlgebras). We use seven publicly available event logs. For each log, we discover two process trees by using different noise threshold values in the discovery algorithm [13] (starting with threshold 10% and increasing with steps of 10% until we discover a different process tree). To reduce time consumption, we sampled 1000 cases per event log.

The results of the experiment are presented in Table 1. In all cases, as expected, we observe that using a 2.5% reduction on the bottleneck yields a better improvement on the overall cycle time of the process, i.e., roughly 2.5 times the 1% reduction. Only in the BPI 2020 Travel Permit data [15], the model impacts the measured cycle time of the identified major bottleneck in the process. Upon inspection, this is the case because the 10%-model incorporates more synchronizations of the bottleneck activity, and hence, more performance measurements, leading to a slightly lower measured activity sojourn time. Furthermore, the aforementioned event log and the BPI 2020 International Declarations [15] event log are the only two event logs in which the process model has an influence on the global performance reduction.

6 Conclusion

In this paper, we presented a foundational framework that allows us to measure the time-based performance of a process, based on historically logged event data. The framework exploits partially ordered alignments (POAs), which are annotated with time-based performance information derived from the data. The use of POAs supports the use of data that records both start and end times of events. The effect of anticipated changes of activity-level performance can be injected into the framework. In our evaluation, we highlight the applicability of our tool using real event data.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4) (2019) 686–705
3. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
4. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Inf. Syst. **34**(3) (2009) 305–327
5. Pourbafrani, M., van Zelst, S.J., van der Aalst, W.M.P.: Scenario-based prediction of business processes using system dynamics. In Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R., eds.: OTM 2019 Conferences - CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019, Proceedings. Volume 11877 of LNCS., Springer (2019) 422–439
6. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: A survey. IEEE Trans. Serv. Comput. **11**(6) (2018) 962–977
7. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2) (2011) 450–475
8. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In Dubois, E., Pohl, K., eds.: CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings. Volume 10253 of LNCS., Springer (2017) 477–492
9. Martin, N., Depaire, B., Caris, A.: The use of process mining in business process simulation model construction - structuring the field. Bus. Inf. Syst. Eng. **58**(1) (2016) 73–87
10. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering colored Petri nets from event logs. Int. J. Softw. Tools Technol. Transf. **10**(1) (2008) 57–74
11. Maruster, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowl. Inf. Syst. **21**(3) (2009) 267–297
12. Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. In: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers. (2014) 75–88
13. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In Lohmann, N., Song, M., Wohed, P., eds.: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers. Volume 171 of LNBIP., Springer (2013) 66–78
14. van Dongen, B.F.: BPI Challenge 2017 (Feb 2017)
15. van Dongen, B.F.: BPI Challenge 2020 (Mar 2020)
16. de Leoni, M.M., Mannhardt, F.: Road traffic fine management process (Feb 2015)
17. Mannhardt, F.: Hospital billing - event log (Aug 2017)